NOTE TO USERS

UNIVERSITÉ DE MONTRÉAL

RECONSTRUCTION 3D DE LA SURFACE EXTERNE DU TRONC HUMAIN POUR LE SUIVI NON EFFRACTIF DES DÉFORMATIONS SCOLIOTIQUES

ALEXIS HÉTU
INSTITUT DE GÉNIE BIOMÉDICAL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE BIOMÉDICAL)
Avril 2004

Your file Votre référence
ISBN: 0-612-91950-1
Our file Notre référence
ISBN: 0-612-91950-1

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

RECONSTRUCTION 3D DE LA SURFACE EXTERNE DU TRONC HUMAIN POUR LE SUIVI NON EFFRACTIF DES DÉFORMATIONS SCOLIOTIQUES

présenté par: HÉTU Alexis
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment acceptée par le jury d'examen constitué de:

M. SAVARD Pierre, Ph.D., président
Mme CHERIET Farida, Ph.D., membre et directeur de recherche
Mme RONSKY Janet, Ph.D., membre et codirecteur de recherche
M. BOUDREAULT Yves, Ph.D., membre

Remerciements

J'aimerais remercier ma directrice de recherche Farida Cheriet autant pour son support moral que financier, pour l'organisation des réunions de l'équipe de recherche, pour avoir pris le temps de remplir de nombreux formulaires et demandes de bourses et pour avoir écrit pour moi plusieurs lettres de recommandation.

Aussi, j'aimerais remercier Janet Ronsky qui a accepté de co-diriger ce mémoire, malgré le fait qu'elle soit à Calgary.

J'aimerais aussi remercier le Fonds québécois de la recherche sur la nature et les technologies (FQRNT, anciennement FCAR ou « Fonds pour la formation des chercheurs et l'aide à la recherche ») et le Fonds de la recherche en santé du Québec (FRSQ) qui sont les organismes subventionnaires qui m'ont fourni l'aide financière durant ces deux années de maîtrise.

De plus, j'aimerais remercier Valérie Pazos pour son aide technique et pour m'avoir régulièrement fourni de nouvelles données nécessaires aux tests et menant aux ajustements à effectuer.

Finalement, j'aimerais remercier Justin Novosad, Vincent Finnerty et Jonathan Boisvert avec qui j'ai eu la chance de travailler. Ces gens m'ont aidé à comprendre plusieurs concepts et théories et m'ont fourni une aide précieuse dans le développement des méthodes décrites dans ce document.

Résumé

La scoliose idiopathique adolescente est une déformation tridimensionnelle complexe des structures anatomiques du tronc. Habituellement, le diagnostic et le suivi de cette maladie sont effectués à partir de radiographies postéro-antérieures et latérales. La mesure souvent reconnue comme le « gold standard » des méthodes de diagnostic de la scoliose est l'angle de Cobb. Cependant, la prise de radiographies expose le patient à des radiations ionisantes qui peuvent nuire à sa santé. Aussi, à cause de la nature tridimensionnelle de la déformation, les méthodes évaluant la déformation tridimensionnelle du tronc sont de plus en plus étudiées. Une de ces méthodes est la topographie de surface.

Jusqu'à très récemment, les topographies de surface ne sont acquises que pour la vue de dos du patient. Une des hypothèses qui motive le groupe de recherche sur les déformations musculo-squelettiques de l'Hôpital Sainte Justine (GRDMS) est que la déformation de la colonne vertébrale entraîne une déformation de la cage thoracique. Pour cette raison, l'acquisition par topographie de surface de tout le tronc pourrait aider à calculer des indices cliniques permettant d'évaluer la sévérité d'une scoliose. Cette nouvelle approche apporte cependant plusieurs nouveaux défis.

L'objectif global de ce projet est de concevoir un outil de reconstruction 3D automatique reproductible et utilisable en clinique.

Le premier objectif spécifique consiste à effectuer la segmentation automatique des images acquises par les appareils de numérisation 3D. Pour l'instant, cette opération est entièrement manuelle. En plus d'être longue et fastidieuse, cette étape ajoute un problème au point de vue de la répétabilité de la méthode. Les variations intra-opérateur et inter-opérateurs sont grandes et la nécessité d'automatiser cette opération devient de

plus en plus critique. La méthode de segmentation automatique proposée est basée sur le « Pseudo-Likelihood Information Criterion ». Cette méthode permet de déterminer le nombre de segments présents dans une image et effectue ensuite la segmentation. Une fois la segmentation terminée, il suffit de choisir les segments représentant la région d'intérêt et de la couper selon quelques critères simples basés sur une connaissance a priori de l'anatomie de la région d'intérêt.

De plus, une nouvelle méthode de représentation d'une image a été proposée. Cette nouvelle représentation permet d'obtenir d'excellents résultats de segmentation. Aussi, une méthode de choix de la région d'intérêt a été élaborée.

Le second objectif spécifique consiste à recaler les différentes surfaces acquises par les numériseurs 3D. Puisque l'acquisition de tout le tronc est nécessaire pour vérifier l'hypothèse de recherche, plusieurs surfaces différentes doivent être acquises puis replacées ensemble de façon à former un tout cohérent, ce qui constitue l'étape du recalage. La méthode actuellement utilisée pour effectuer ce recalage est la numérisation d'une toile placée dans le champ de vision de tous les numériseurs, ce qui permet de calculer les matrices de transformation reliant les différents numériseurs. Cette étape doit être faite avant les acquisitions de données afin de pouvoir reconstruire tout le tronc en 3D. Le problème est que non-seulement cette opération doit être faite par un opérateur, mais la configuration de numériseurs utilisée introduit des erreurs de recalage dues au fait que l'axe de projection n'est pas toujours perpendiculaire à la toile.

Pour effectuer le recalage, la méthode appelée « Iterative Point Matching » (IPM) a été utilisée. Cette méthode permet essentiellement de raffiner un recalage dont on connaît déjà une bonne approximation initiale. La solution trouvée est la meilleure au sens des moindres carrés de la distance séparant les points de la première surface aux points de la seconde surface. Une solution alternative permettant d'éliminer une partie

des points avant d'appliquer l'algorithme IPM a été proposée. Puisqu'une approximation initiale est nécessaire, une seconde méthode, appelée « point fingerprints » a été utilisée. Cette méthode se sert des déformations locales de chaque surface afin de déterminer quels points de chaque surface sont correspondants. Les points présentant le plus de déformation locale sont conservés comme points d'intérêt puis appariés par une corrélation croisée. Plusieurs adaptations ont été effectuées afin que l'algorithme soit plus robuste. L'ajout d'un nouveau type de corrélation et de nouvelles méthodes d'appariements de points jouent un rôle important dans la qualité des résultats obtenus.

L'algorithme de sélection automatique de région d'intérêt a démontré qu'il était possible de déterminer de manière précise le contour de la région d'intérêt dans plus de 85% des cas, ce qui est déjà un grand pas en avant du point de vue de la répétabilité de la procédure. Avec l'ajout de conditions standardisées, l'algorithme sera parfaitement applicable à l'application courante.

Le recalage des surfaces par IPM est fonctionnel et donne d'excellents résultats. La méthode des *point fingerprints* a démontré qu'avec quelques adaptations, il est possible d'apparier avec précision des points d'intérêt provenant de deux surfaces partiellement correspondantes. Cette méthode est cependant sensible au bruit et n'est pas utilisable sur des surfaces trop bruitées, ce qui est normal puisque le bruit change l'allure des déformations locales.

Les objectifs de ce mémoire ont été atteints dans la mesure où un outil de reconstruction 3D automatisé a été développé et est fonctionnel pour l'utilisation en clinique. Il ne reste qu'à trouver un moyen de diminuer le bruit sur les surfaces acquises et les algorithmes de détection automatique de la région d'intérêt et de recalage de surfaces pourront être utilisés avec confiance.

À l'avenir, la standardisation de la procédure d'acquisition devrait contenir les conditions suivantes :

- Standardisation du positionnement des patients (bras droits, écartés à environ 45 degrés du corps est la position préférée pour l'instant)
- Pas de cheveux dans le cou
- Pas de soutien-gorge
- La présence de pantalons ou d'une jupe favorise la détection du bas du dos
- Arrière-plan monochromatique (écran bleu ou écran vert, par exemple)
- Sources de lumières provenant exclusivement des caméras autant que possible

Finalement, dans un avenir proche, il serait intéressant de développer un filtre permettant l'obtention de données de meilleure qualité. Ceci permettrait non seulement l'utilisation régulière des algorithmes développés dans ce mémoire, mais aussi l'utilisation des données dans plusieurs études futures.

Abstract

Adolescent idiopathic scoliosis is a three-dimensional deformation of the spine and rib cage. Usually, the diagnosis and follow-up of this disease are done with postero-anterior and lateral radiographs. The Cobb angle is the current gold standard for assessing scoliosis. The problem with radiographs is that the patient has to be exposed to ionizing radiation, which can be damaging to a patient's health. Also, because of the three-dimensional nature of the deformation, three-dimensional methods for assessing scoliosis are of greater interest and are the subject of an increasing number of studies. One of those methods is surface topography.

Until recently, only a surface topography of the back of the patient was used in order to try to detect a scoliosis. In our case, an important research hypothesis is that the deformation of the spine causes a deformation in the rib cage. Because of that, a surface topography of the whole trunk could help to assess scoliosis more accurately. This new approach also brings new challenges.

The main goal of this research is to conceive a 3D reconstruction tool that is reproducible and usable in a clinical context.

The first step is to perform an automatic segmentation on the images received from the 3D digitizers. Until now, this operation was entirely manual. Apart from being a long and tedious process, the manual segmentation adds a repeatability problem. The inter-operator variability is quite large and the need for an automatic solution is obvious. The automatic segmentation method chosen for this task is called the "Pseudo-Likelihood Information Criterion". This method finds the number of different regions in an image and performs the image segmentation. Once the segmentation is done, a simple set of rules allows us to find which regions are the right ones and how to remove unwanted parts of those regions. To this end, this paper proposes a new image representation that provides excellent segmentation results. Also, a region selection method based on the anatomical knowledge of the region of interest has been proposed.

The second step is to register the surfaces received from the 3D digitizers. Since a 3D model of the whole trunk is needed in order to verify the research hypothesis, many different surfaces must be digitized and reassembled so that they make sense as a whole. This process is called registration. The currently used registration method is the digitization of a large sheet visible to all digitizers, which is then used to calculate the transformation matrices needed to register the surfaces acquired by the different digitizers. This step is done before the surface acquisitions for a patient. The problem is that not only must an operator perform this operation, but it introduces errors in the final registration because the sheet is not perpendicular to the projection axis.

Instead of manual registration, the chosen registration method is called "Iterative Point Matching" (IPM). This method refines an already approximated registration. The solution given minimizes the least squares of the distance between the points of each surface.

Since an approximation of the registration must be known, another method, called "point fingerprints" must be used. This method uses the local deformations of a surface to find which points correspond to the same point in reality. The points having the largest local deformations are considered to be interest points which will then be matched with a cross correlation. Several adaptations to the original algorithm were made so it would be more robust. The addition of a new type of correlation and new methods of point matching play an important part in the quality of the obtained results.

The algorithm for automatic selection of an area of interest showed that it was possible to find in a precise way the contour of the area of interest in more than 85% of the cases used for this study, which is already a great step towards the repeatability of the procedure. With the addition of standardized conditions, the algorithm will be perfectly applicable to the current application.

Surface registration by IPM is functional and gives excellent results. The point fingerprints method showed that, with some adaptations, it is possible to pair, with good precision, points of interest coming from two partially corresponding surfaces. However, this method is sensitive to noise and is not usable on very noisy surfaces, which is normal since noise significantly changes the shape of local deformations.

The goals of this research were achieved and a tool for automatically reconstructing 3D surfaces was developed and is clinically usable. If a way of decreasing the noise on acquired surfaces is found, the algorithms of automatic detection of the area of interest and surface registration could be used with confidence. In the future, the standardization of the acquisition procedure should contain the following conditions:

- Standardization of the positioning of the patients (arms straight, apart from the body with approximately a 45 degrees angle is the preferred position for now)
- No hair over the neck
- No bra
- The presence of trousers or a skirt helps with the lower back detection
- A monochromatic background (blue screen or green screen, for example)
- Using only the sources of lights from the cameras as much as possible

Finally, in the near future, it would be interesting to develop a filter that would provide better data quality. This would allow not only the use of the algorithms developed in this paper, but also the use of the data in several future studies.

Table des matières

Liste des tableaux

Liste des figures

Liste des annexes

Liste des sigles et abréviations

A.D.A.M.	Animated Dissection of Anatomy for Medicine
BIC	Bayesian Information Criterion
FDP	fonction de densité de probabilité
ICM	Iterative conditional modes
IPM	Iterative Point Matching
MAP	maximum *a posteriori*
MMAP	maximum marginal *a posteriori*
MV	maximum de vraisemblance
PCRPCC	Bureau de la protection contre les rayonnements des produits cliniques et de consommation
PET	Positron Emission Tomography (voir TEP)
PF	Point Fingerprints
PLIC	Pseudo-likelihood information criterion
SRS	Scoliosis Research Society
TEP	Tomographie par Emission de Positrons
TS	Topographie de surface
VRML	Virtual Reality Modeling Language

Chapitre 1 : Introduction

La scoliose est une déformation tridimensionnelle complexe des structures anatomiques du tronc (*i.e.* colonne vertébrale, cage thoracique). La figure 1.1 présente un exemple de déformation scoliotique. La prévalence de déformations graves est de 2 à 3% parmi les patients atteints âgés entre 10 et 16 ans. La scoliose atteint 2 fois plus de filles que de garçons, mais 8 fois plus de filles que de garçons ont des déformations graves. Aussi, 85% à 90% des scolioses sont idiopathiques (*i.e.* sans cause connue). (Ces statistiques peuvent varier légèrement selon la source. Voir Scoliosis Research Society 2003, REAMY et SLAKEY 2001 et Family Practice Notebook 2000).

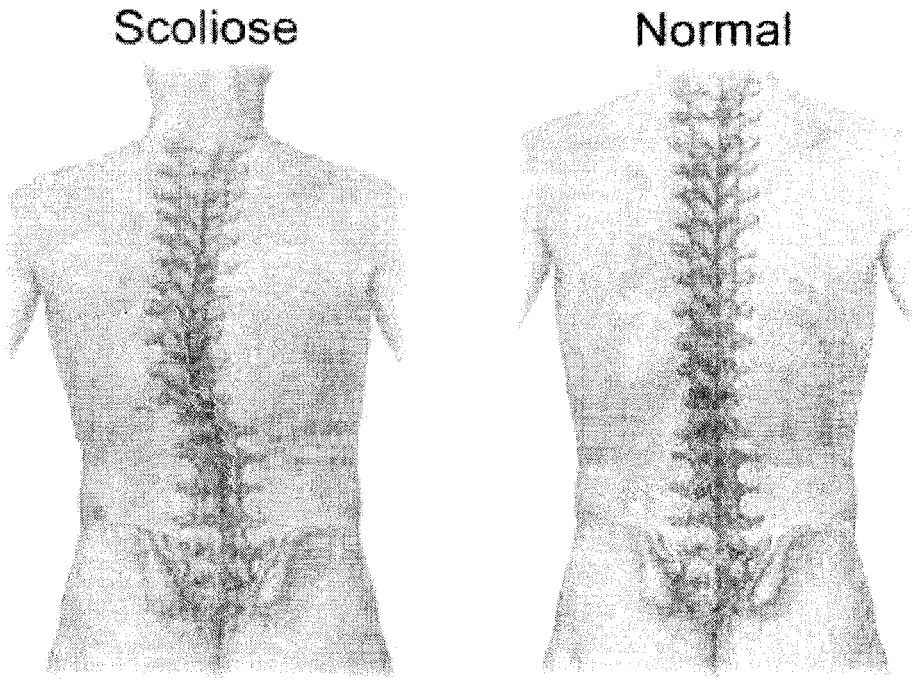


Figure 1.1: Colonnes vertébrales avec et sans scoliose (A.D.A.M. 2003)

La scoliose engendre plusieurs problèmes comme des maux de dos (Joncas et coll. 1996) et des problèmes psychologiques, par exemple mauvaise estime de soi dû à une déformation visible du dos (Goldberg et coll. 1994, Payne et coll. 1997). À un stade

avancé, la maladie peut nuire à la réalisation d'activités physiques, ce qui en soi engendre d'autres problèmes physiques et psychologiques (Goldberg, 1994) et altérer les fonctions pulmonaires et cardiaques (Weinstein et coll. 1981).

Habituellement, le diagnostic et le suivi de cette maladie sont effectués à partir de radiographies postéro-antérieures et latérales. La mesure souvent reconnue comme le « gold standard » des méthodes de diagnostic de la scoliose est l'angle de Cobb tel qu'illustré à la figure 1.2.

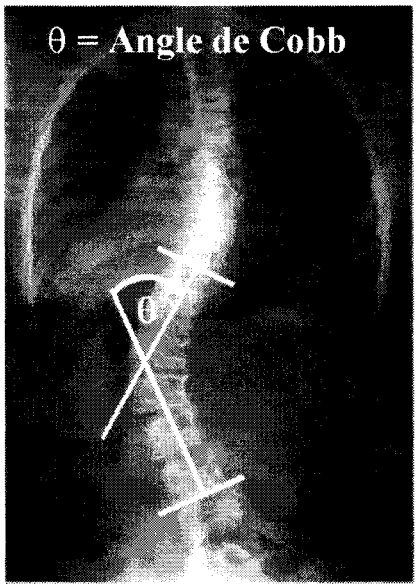


Figure 1.2: Angle de Cobb (University of California, San Francisco 2003)

Il existe aussi des méthodes d'évaluation 3D de la scoliose comme la reconstruction 3D à partir des radiographies. Certains développements récents dans le domaine ont permis « d'introduire une nouvelle classe de techniques de reconstruction 3D qui permettent de produire des modèles 3D des structures osseuses en utilisant de l'équipement et des protocoles radiologiques standard » ou à partir de plusieurs radiographies ou même à partir d'une seule radiographie. Dans le second cas, « la technique est basée sur le recalage 3D/2D de modèles de vertèbres connues *a priori* » (Novosad 2002).

Selon Santé Canada :

Les rayons X sont des rayonnements électromagnétiques, tout comme la lumière visible, les rayons ultraviolets et les micro-ondes, à des niveaux d'énergie différents. Ce sont toutefois les rayons X qui possèdent le plus d'énergie. Ils peuvent ioniser (charger électriquement) le matériel qu'ils traversent, entraînant des lésions des cellules ou de l'ADN dans la matière vivante (Bureau de la protection contre les rayonnements des produits cliniques et de consommation [PCRPCC] 2002a).

Toujours selon Santé Canada :

Aucune technique entraînant l'irradiation ne devrait être adoptée si elle n'engendre pas pour les individus exposés ou la société assez d'avantages pour compenser le préjudice radiologique qu'elle cause (PCRPCC 2002b).

[PCRPCC] 2002b fournit aussi un modèle de gestion afin de minimiser les risques du cancer radio-induit.

Cependant, ces risques sont toujours présents et révèlent le côté effractif de la radiographie. Pour ces raisons, en clinique, on limite le suivi des patients à une visite à tous les six mois. Cela explique pourquoi, dans le cadre des recherches sur les déformations musculo-squelettiques du tronc, il est souhaitable de pouvoir fournir une alternative non-effractive à la radiographie afin de pouvoir effectuer un suivi régulier et fréquent de l'évolution de la maladie chez un patient.

Ce suivi est de haute importance car il sert à guider le traitement de la maladie. Si la maladie progresse trop vite, il faudra procéder à un traitement. Dans un premier temps, le port d'un corset peut être utilisé pour corriger la déformation scoliotique du patient. Si la maladie est trop grave, il faudra procéder à une chirurgie. Dans certains cas, il est possible que la chirurgie ait pu être évitée si le suivi avait été effectué à des intervalles de temps plus courts. La chirurgie est une procédure pénible et coûteuse durant laquelle on doit insérer des vis dans certaines vertèbres de la colonne vertébrale (Weinstein 1994). L'intérêt d'un suivi plus fréquent de la maladie devient alors assez clair.

Le projet réalisé dans ce mémoire s'inscrit dans le cadre des méthodes d'évaluation et de suivi non effractif des déformations scoliotiques. Ce mémoire a été subdivisé en plusieurs chapitres et sous-chapitres. La figure 1.3 de la page suivante présente le plan du mémoire ainsi que la progression à suivre au cours des chapitres.

Au chapitre 2 intitulé « Revue de la littérature », les méthodes et technologies existantes pour l'évaluation non effractive de la scoliose seront exposées. Ensuite, la technologie adoptée au sein de l'équipe ainsi que les limites de cette technologie seront présentées. Enfin, le reste du chapitre présente un état de l'art sur les différentes techniques de segmentation et de recalage d'images utilisées pour la réalisation de ce projet, suivi des objectifs généraux et spécifiques.

Au chapitre 3 intitulé « Méthodologie », le développement des méthodes de segmentation d'image et de recalage sera expliqué. Aussi, la méthode de validation des résultats des algorithmes sera présentée.

Au chapitre 4 intitulé « Résultats et Discussion », le choix des paramètres du modèle adopté sera expliqué et les résultats de la détection de la région d'intérêt et ceux du recalage 3D seront présentés puis discutés.

Au chapitre 5, la conclusion sur la détection de la région d'intérêt et celle sur le recalage 3D seront présentées suivi d'une conclusion générale sur l'ensemble du mémoire où les perspectives seront discutées et des recommandations seront formulées.

Bonne lecture !

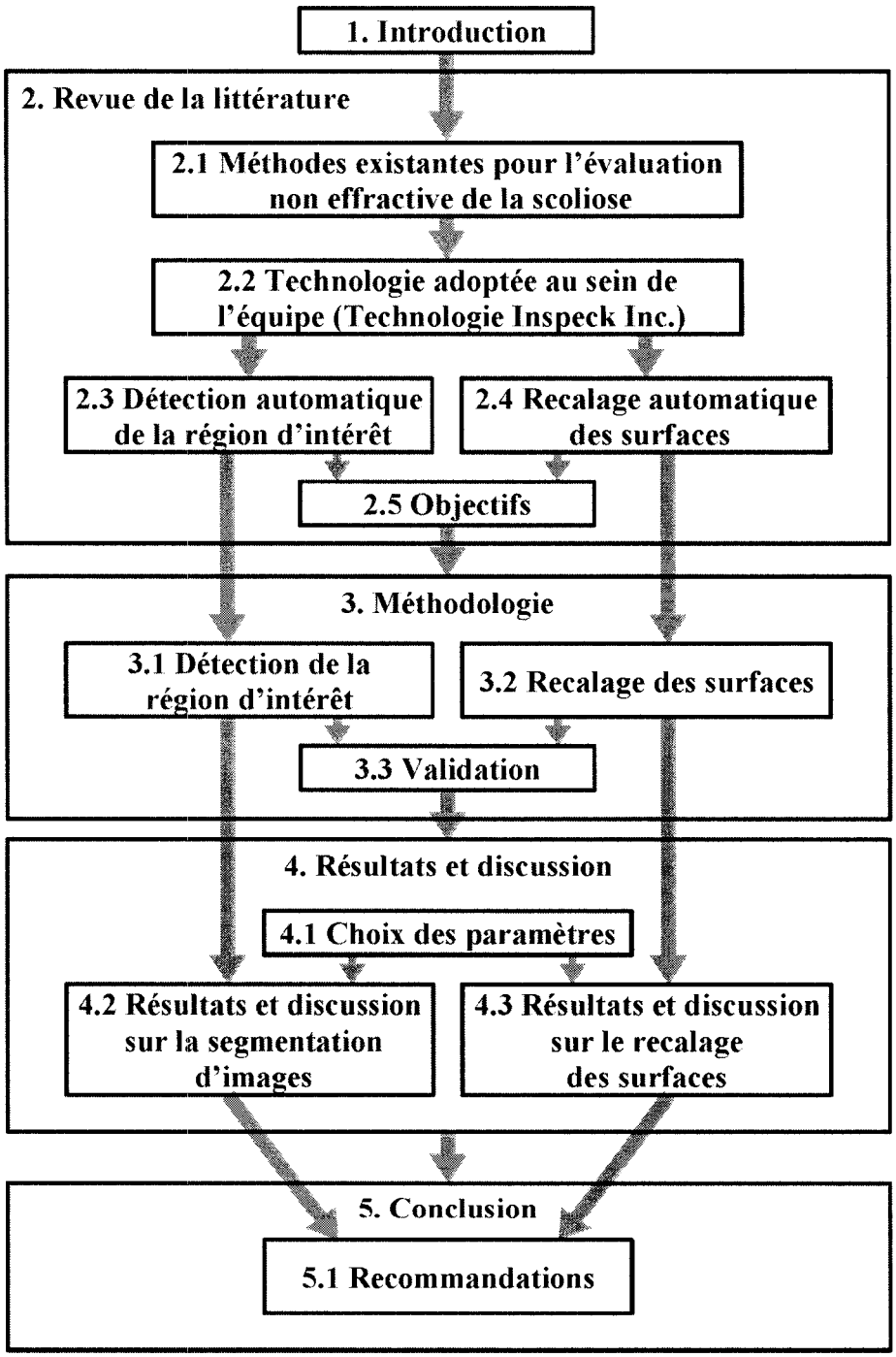


Figure 1.3: Plan du mémoire

Chapitre 2 : Revue de la littérature

Le contenu de ce chapitre sera divisé en quatre sections principales. Tout d'abord, les méthodes et technologies existantes pour l'évaluation non effractive de la scoliose seront exposées. En second lieu, la technologie adoptée au sein de l'équipe ainsi que les limites de cette technologie seront présentées. Enfin, les deux dernières sections du chapitre présentent un état de l'art sur les différentes techniques de segmentation (troisième section) et de recalage d'images (quatrième et dernière section) utilisées pour la réalisation de ce projet d'image.

2.1 Méthodes existantes pour l'évaluation non effractive de la scoliose

Jusqu'à maintenant, beaucoup d'efforts ont été mis dans l'élaboration d'une méthode à la fois fiable et non-effractive d'évaluation des déformations scoliotiques. Les méthodes actuelles ne répondent malheureusement pas encore à tous les besoins des cliniciens. Les méthodes ayant connu une certaine popularité dans le passé seront donc présentées ici par ordre de sophistication croissant. Les limites de chacune des méthodes seront aussi soulignées.

2.1.1 Scoliomètre

Tout d'abord, le scoliomètre est un appareil servant à mesurer les distorsions du torse. La procédure se déroule ainsi :

- Le patient se penche par en avant avec les bras pendants et les paumes collées ensemble jusqu'à ce qu'une courbe puisse être observée dans la région thoracique.
- Le scoliomètre est placé sur le dos et mesure le point le plus haut de la courbure thoracique (voir figure 2.2).

- Le patient continue à se pencher jusqu'à ce qu'une courbe apparaisse dans la partie lombaire du dos et on mesure le point le plus haut de la courbure lombaire.
- On répète la procédure deux fois et le patient revient en position verticale entre les deux mesures.
- Si on détecte la présence d'une déformation, une radiographie doit être prise afin de mesurer l'ampleur de la déformation.

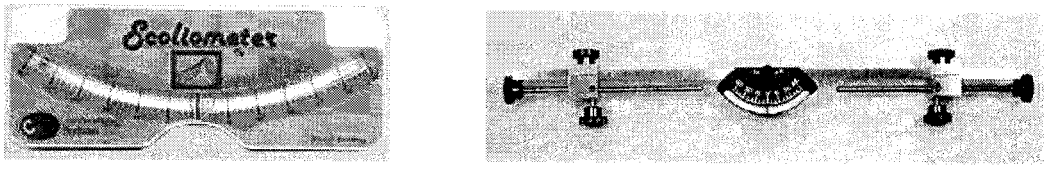


Figure 2.1: Exemples de scoliomètres

Figure 2.2: Exemple d'utilisation du scoliomètre

Le scoliomètre aurait pu servir à faire des dépistages à grande échelle de la scoliose, mais, en pratique, plus de la moitié des enfants ayant des déformations mineures ou absentes ont indiqué une déformation sur le scoliomètre. Puisqu'il est important de minimiser l'utilisation des rayons X, le nombre de faux positifs est trop grand pour permettre ainsi une utilisation à grande échelle. De plus, le scoliomètre a été jugé trop imprécis pour guider le traitement de la scoliose (A.D.A.M. 2002).

2.1.2 Test par flexion antérieure

Il existe aussi un test permettant l'analyse de la gibbosité (bosse dans le dos du patient). Cette analyse se fait en flexion antérieure du rachis (penché en avant) et permet de déceler à 1 ou 2 mm près la valeur de la gibbosité (Bruandet 1996). Ce test n'est cependant pas sensible aux anomalies présentes dans la région lombaire, ce qui fait qu'environ 15% des cas de scoliose ne sont pas détectés (A.D.A.M. 2002). Ceci explique pourquoi on recommande souvent de ne pas utiliser uniquement ce test pour effectuer le dépistage de la scoliose.

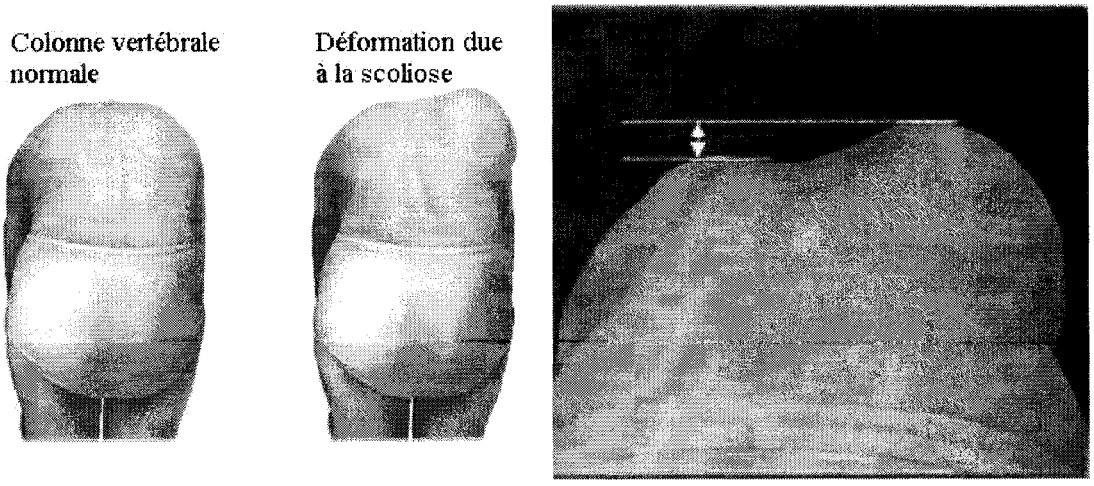


Figure 2.3: Test par flexion antérieure (A.D.A.M. 2003)

2.1.3 Patrons de Moiré

Définissons d'abord ce qu'est la topographie de surface (TS). Elle correspond à une image tridimensionnelle obtenue habituellement à l'aide de caméras CCD (charged coupled devices). Il est possible de l'utiliser à l'échelle microscopique (étude de la surface d'une fracture, inspection industrielle) autant que macroscopique (océans). Dans le cas présent, la topographie de surface utilisée est celle du dos du patient.

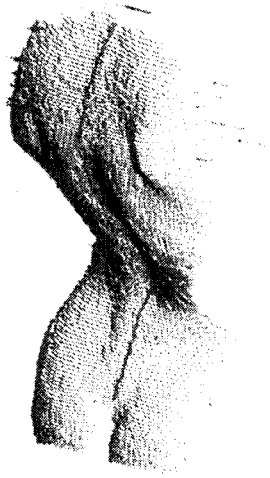


Figure 2.4: Topographie de surface du dos avec épine dorsale en évidence (tiré de *Archives on Disease in Childhood* http://adc.bmjjournals.com/)

Initialement, des recherches ont été effectuées sur les patrons de Moiré. Cette méthode utilise de ce que nous appellerons plus tard de la « lumière structurée » pour calculer la forme tridimensionnelle d'un objet grâce à la déformation de cette lumière structurée due au relief de l'objet. La lumière structurée est un ensemble connu de patrons lumineux (par exemple un ensemble de lignes horizontales ou verticales).

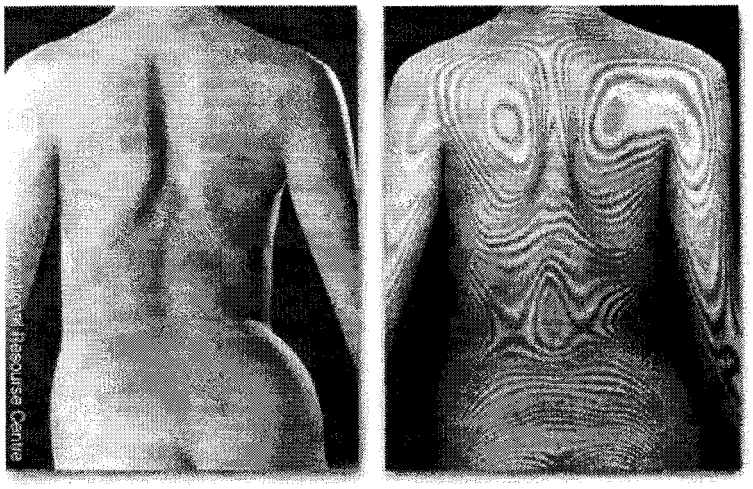


Figure 2.5: Utilisation des patrons de Moiré pour la détection de la scoliose (tiré de *Educational Resource Centre* http://www.rch.unimelb.edu.au/erc)

Suzuki et coll. 1992 et Karachalios 1999 ont conclu que :

- dans un cadre bien défini, la topographie de surface (TS) Moiré permet un suivi intensif avec moins de radiations;
- la technique de Moiré ne devrait pas être utilisée seule pour la détection de la scoliose à cause du trop grand nombre de faux positifs;
- la technique de Moiré détecte pratiquement tous les cas de scoliose.

Mais le plus important est qu'il n'est pas possible de conclure sur l'impact économique à long terme de la détection préventive de la scoliose dans les écoles à cause du nombre trop élevé de personnes qui doivent consulter un médecin inutilement par rapport aux coûts du petit nombre de personnes qui auront besoin d'une réelle assistance médicale.

Il est à noter que la raison pour laquelle le nombre de faux positifs est élevé est que la formation des patrons de lumière dans le dos du patient est fortement influencée par l'angle entre le dos du patient et la caméra. Ainsi, une légère rotation de la part du patient change significativement la forme des patrons de lumière obtenus.

2.1.4 ISIS (integrated shape imaging system)

Tout comme les patrons de Moiré, la méthode ISIS (Turner-Smith 1988, Turner-Smith et coll. 1988) se sert aussi de lumière structurée pour déterminer la forme tridimensionnelle d'un objet.

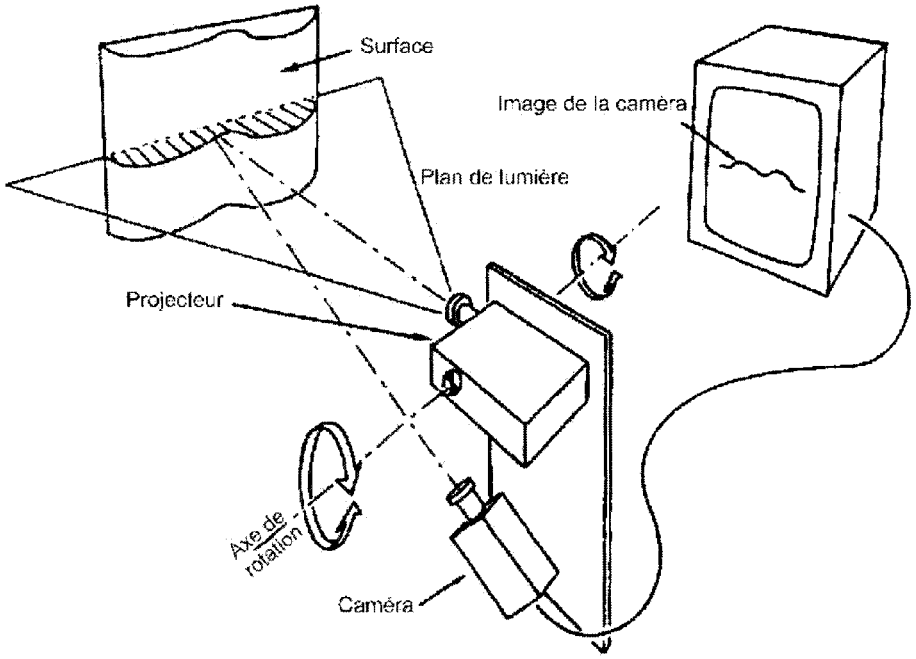


Figure 2.6: Principe du système ISIS

Un projecteur envoie un plan de lumière capté par une caméra placée sous le projecteur. L'avantage d'utiliser un patron lumineux aussi simple est que les coordonnées 3D sont faciles à calculer. Le désavantage est que l'on doit balayer toute la surface. Pour numériser tout le dos d'un patient, cela peut prendre de 1 à 2 secondes. Les légers mouvements du patient durant cette période peuvent causer des erreurs. De plus une calibration de l'appareil est nécessaire et peut ajouter à l'erreur. La résolution de l'appareil est de l'ordre de ± 1.5 mm et la précision de l'appareil, pour un volume de 400mm × 500mm × 300mm, est de moins de 3mm (Turner-Smith et coll. 1988).

2.1.5 Topographie de surface : numériseurs BIRIS

À l'Université de Calgary, une numérisation du tronc complet a été effectuée à l'aide de quatre numériseurs laser BIRIS. Cette numérisation prends 15 secondes à s'effectuer. Ce temps est dû au balayage vertical et limite la résolution verticale à 6.7mm (Jaremko 2001). Les mouvements du patient durant ces 15 secondes peuvent causer des erreurs qui seront désignées plus tard sous le nom « flou de mouvement ».

2.1.6 Topographie de surface Quantec

Récemment, des études ont porté sur la méthode de topographie de surface Quantec. Cette méthode n'utilise qu'une source de lumière, une caméra et un ordinateur. La figure 2.4 est un exemple de ce type de topographie de surface. Goldberg et coll. 2001 et Liu et coll. 2001 ont déduit de leurs recherches que :

- les mesures topographiques n'ont pas une forte dépendance à l'angle de Cobb;
- l'angle de Cobb ne peut pas à lui seul décrire toute la surface déformée, mais il existe une corrélation entre l'angle de Cobb et la surface;
- TS Quantec permet de faire la différence entre les patients avec un angle de Cobb $< 10°$ et ceux ayant un angle de Cobb $> 20°$. Ceci est cliniquement important.
- chez des patients souffrant d'une scoliose mineure une progression de l'angle de Cobb significative est détectable avec TS Quantec un an avant qu'elle ne se manifeste dans la radiographie.

Il est important de se rappeler que les patients atteints de la scoliose se plaignent de la déformation, et non de l'angle de Cobb et qu'une reconstruction 3D à partir de la TS décrit mieux cette déformation. Donc, la topographie de surface réduit l'écart entre ce que l'observateur voit et ce que le clinicien peut mesurer.

La technique est donc prometteuse, mais sa précision doit être améliorée pour qu'elle devienne utilisable en clinique.

2.2 Technologie adoptée au sein de l'équipe (Technologie InSpeck Inc.)

La topographie de surface étant une nouvelle méthode prometteuse de diagnostic et/ou de suivi de la scoliose, les chercheurs de ce domaine cités plus tôt tentent d'établir de nouveaux indices cliniques permettant l'évaluation de la déformation de la colonne vertébrale à partir de la surface externe du tronc. Quoique la topographie de surface ne décèle pas directement la configuration des structures anatomiques internes, elle fournit toutefois suffisamment d'information pour faire un suivi adéquat de la progression de la déformation (Pazos 2002). Jusqu'à maintenant, la plupart des études ont porté sur l'étude de la surface du dos uniquement. Puisqu'il est probable que la déformation de la colonne vertébrale entraîne aussi la déformation de la cage thoracique, l'étude complète du tronc pourrait favoriser la qualité du diagnostic ainsi que l'élaboration de nouveaux indices cliniques.

Pour cette raison, le système utilisé pour l'instant à la clinique de scoliose de l'Hôpital Sainte-Justine est composé de 4 caméras 3D (numériseurs) de InSpeck Inc. Les caméras employées pour l'acquisition des images de topographie de surface à l'hôpital Sainte-Justine en clinique de scoliose sont des 3D Capturor de la compagnie InSpeck inc. de Montréal.

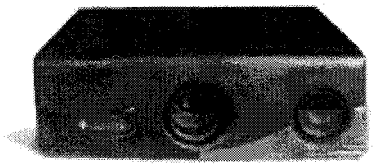


Figure 2.7: Le 3D Capturor de InSpeck inc.

L'acquisition des images est contrôlée par les logiciels FAPS et EM de InSpeck Inc. FAPS est principalement utilisé au moment de l'acquisition des images afin d'acquérir les données brutes alors que EM est un logiciel utilisé pour effectuer des traitements sur les surfaces tels que le recalage (cette étape requiert toutefois une opération manuelle, comme nous le verrons plus tard).

Sur la figure 2.8, on peut voir un exemple de ce montage. La raison pour laquelle certains numériseurs faisant face à la personne sont de biais est pour obtenir toute l'information possible sur la cage thoracique et que les parties latérales du tronc ne sont pas visibles par les deux autres numériseurs.

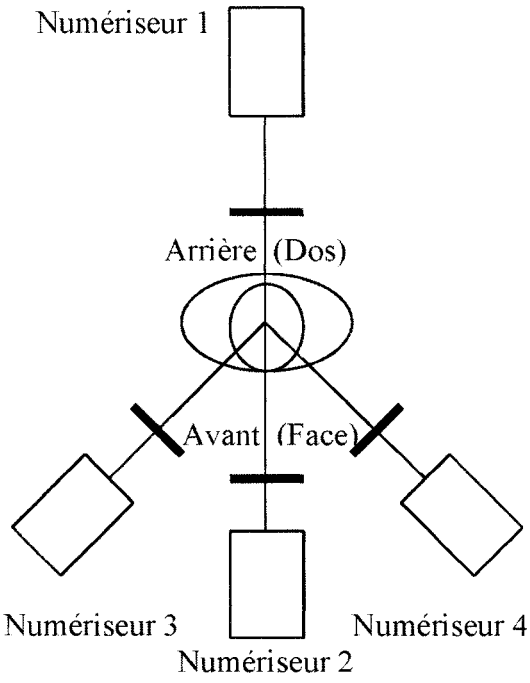


Figure 2.8: Exemple de montage de numériseurs InSpeck Inc.

2.2.1 Principes de fonctionnement

Les mesures optiques de formes tridimensionnelles fournissent des données absolues de la géométrie 3D des objets et ces mesures se doivent d'être indépendantes de la réflexion (diffuse ou spéculaire) de la surface, de la distance qu'il y a entre l'objet et le capteur et de la luminosité ambiante. La forme et les dimensions physiques fournies par le capteur doivent donc être indépendantes de la translation, rotation ou illumination de l'objet (Jähne, Haußecker 2000).

2.2.1.1 Principes physiques

Les différents capteurs peuvent être classés selon trois (3) principes physiques différents : la triangulation, la mesure du temps de vol (incluant l'interférométrie à large bande) et l'interférométrie classique. L'incertitude des différentes techniques varie entre environ un nanomètre jusqu'à quelques millimètres.

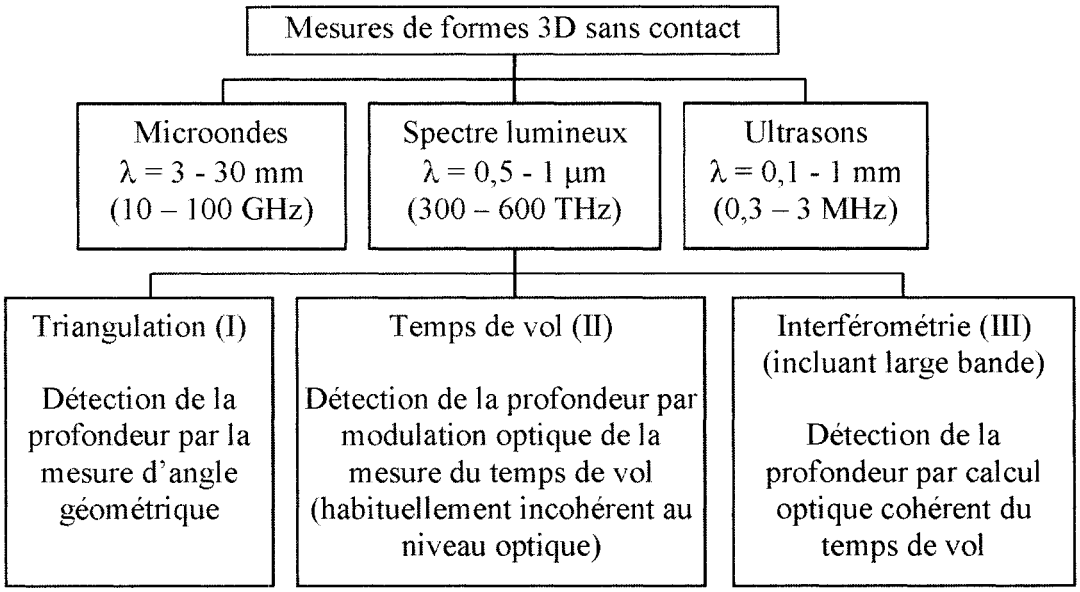


Figure 2.9: Principes de mesures de formes 3D sans contact

(Jähne, Haußecker 2000)

Pour ces trois différents principes, la mesure de l'incertitude change selon la distance z entre l'objet et le capteur ainsi :

type I : $\delta z \propto z^2$, type II : $\delta z \propto z^0$, type III : $\delta z \propto z^{-1}$

Les patrons de Moiré, ISIS et les technologies Quantec et InSpeck utilisent toutes le principe de triangulation. Il existe toutefois d'autres méthodes pouvant utiliser d'autres principes de fonctionnement, mais il y a très peu d'utilisations récentes de ces méthodes pour effectuer des topographies de surface, principalement pour numériser un objet s'apparentant à un tronc humain.

2.2.1.2 Image de profondeur

Avant de détailler les principes de fonctionnement, il est important de comprendre ce que l'on désire obtenir de ces différentes méthodes.

Le résultat attendu pour la capture d'une topographie de surface est la distance à laquelle se situe chaque point de la scène observée par rapport à l'appareil de capture. Cet ensemble d'informations spatiales, fournies sous forme de coordonnées polaires ou cartésiennes, forme une image de profondeur. Aussi, il est maintenant possible d'obtenir la couleur de l'objet grâce à l'application d'une texture capturée par une caméra CCD. Voici un exemple d'acquisition d'une image de profondeur simple sans texture obtenue par interférométrie.

Tout d'abord, un numériseur projette successivement quatre patrons de franges sur la cible. Chaque patron est décalé de un quart de phase par rapport au patron précédent.

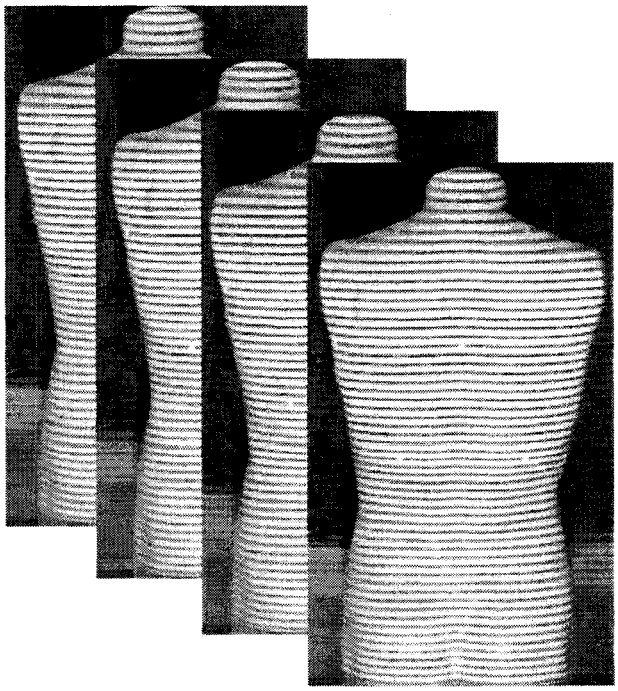


Figure 2.10: Images avec franges

À la figure 2.10, on voit un exemple de ces patrons de franges. Il n'est pas évident de déceler le déphasage, mais en regardant bien le haut du cou, on voit que le patron projeté semble descendre en partant de l'image arrière jusqu'à l'image avant. À partir de ces images, le logiciel FAPS de InSpeck construit une image de phase. La représentation de l'image de phase est en tons de gris (nombres 0-255). Dans le traitement d'une image de phase, il faut toujours tenir compte de la nature cyclique de la phase. Donc, il s'agit d'une représentation modulo 256, tel qu'illustré dans la figure 2.11.

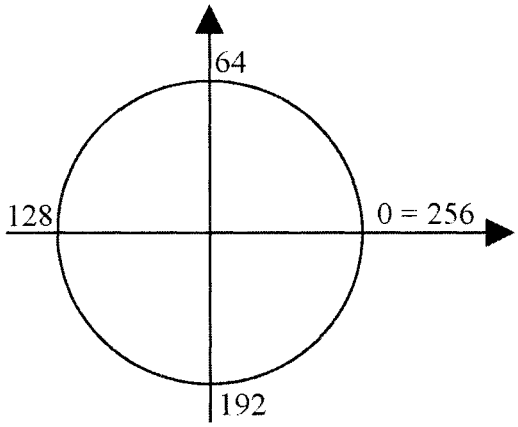


Figure 2.11: Illustration de la progression cyclique de la phase

À l'aide de la figure 2.12, nous pouvons apprécier plusieurs aspects de ces images. Tout d'abord, la continuité entre le noir (0) et le blanc (255) est facilement observable grâce au passage abrupte du noir au blanc visible dans le dos. Aussi, la toile de fond à l'arrière-plan est beaucoup plus bruitée que le dos à l'avant-plan, démontrant ainsi que l'incertitude quant à la distance des objets augmente si l'objet se trouve plus loin du capteur. Il est à noter que l'appareil ayant produit les images d'interférométrie qui ont été utilisées afin de produire cette image de phase utilise à la fois la triangulation et l'interférométrie, puisque seules les bandes horizontales (l'orientation des bandes n'a théoriquement pas d'importance) de lumière sont projetées sur l'objet, pour ensuite être capturées par triangulation puis traitées de manière logicielle. On appelle aussi cette technique la « triangulation active ».

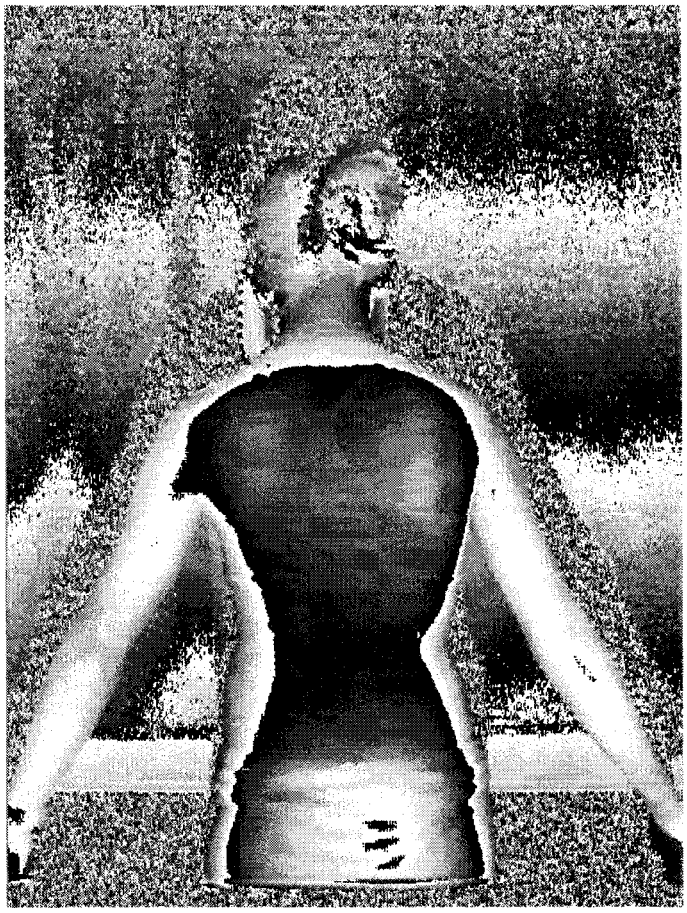


Figure 2.12: Image de phase du dos d'une patiente

Une fois cette image obtenue, il faut « dérouler » la phase pour obtenir la profondeur relative de chaque point de l'image au lieu d'une représentation 0-255 et le logiciel FAPS de InSpeck effectue cette opération. Essentiellement, la phase ne peut être détectée qu'entre $-\pi$ et π à cause de sa nature cyclique, mais il est possible que le déphasage réel soit beaucoup plus grand. Le déroulement de phase est le processus de reconstruction de l'image de déphasage réel à partir de l'image de phase « non déroulée ». L'opération consiste à ajouter et soustraire des multiples de 2π aux endroits appropriés pour rendre l'image de phase aussi lisse que possible. Pour convertir une image en phase en image de déphasage réel, il faut toujours procéder à un déroulement de phase.

2.2.1.3 Triangulation

Puisque la plupart des technologies connues utilise la triangulation, il est nécessaire de présenter cette notion de façon plus détaillée. Ceci permet de mieux situer la méthode de triangulation utilisée par InSpeck par rapport aux autres méthodes.

La triangulation est la technique optique la plus répandue pour la mesure de surfaces dans un espace tridimensionnel. La figure 2.13 présente la classification des diverses méthodes existantes.

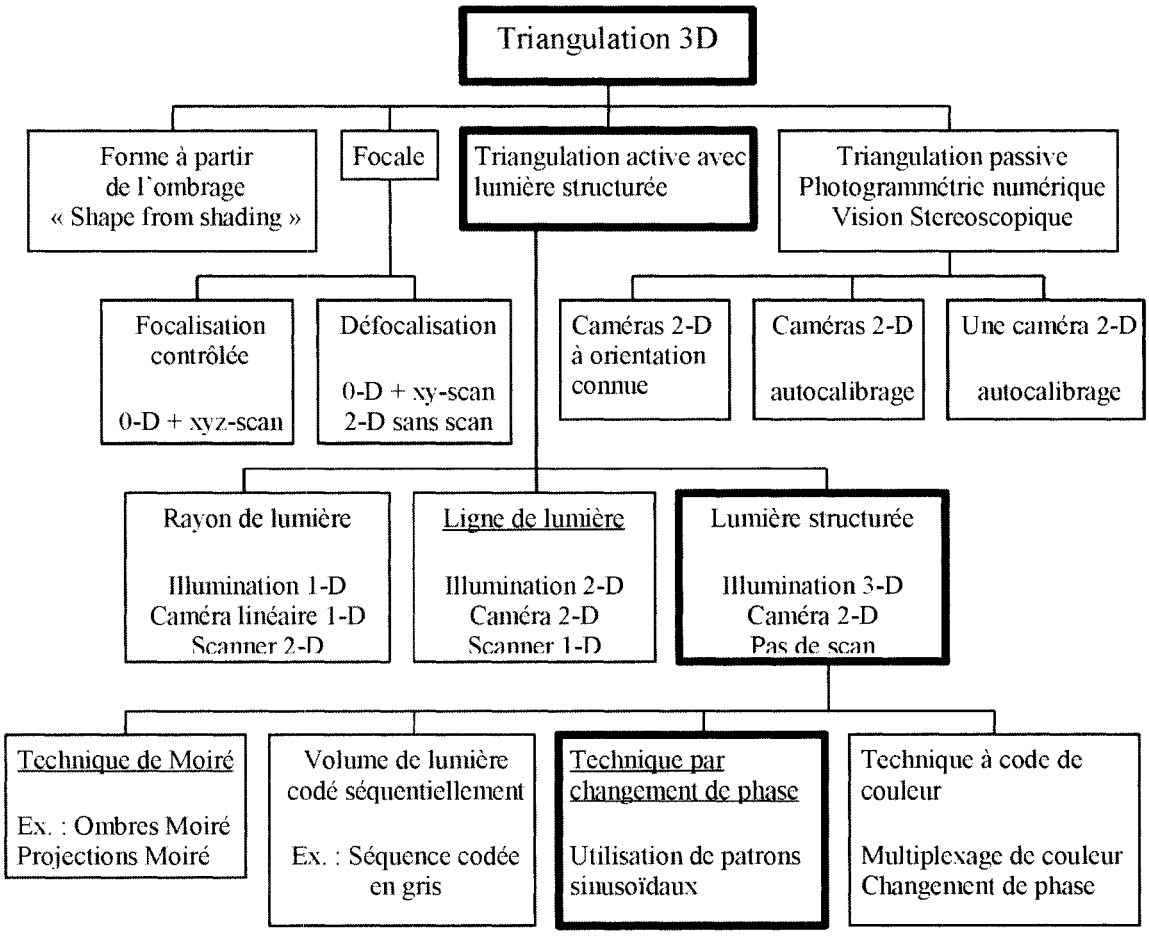


Figure 2.13: Hiérarchie des principes de triangulation les plus importants

(Inspiré de Jähne, Haußecker 2000)

De façon générale, la triangulation consiste à éclairer un point de la scène, puis de détecter la position de cette lumière avec un capteur qui se trouve à un endroit différent de la source de lumière. Connaissant l'angle de projection de la lumière, l'angle de détection et la position de la source et du capteur, il est alors possible de déterminer la position tridimensionnelle du point éclairé. En pratique, on n'éclaire pas toujours avec un point, parfois il est préférable d'utiliser une ligne de lumière ou encore un ensemble de franges lumineuses. D'autres techniques utilisent aussi la lumière ambiante.

Les techniques utilisées dans les méthodes de diagnostic précédentes sont soulignées dans la figure 2.13 et la technique spécifique à InSpeck est encadrée en gras. On y voit la technique de Moiré, l'utilisation d'une ligne de lumière (ISIS et technologie Quantec) et l'utilisation de lumière structurée avec changement de phase (technologie InSpeck). Toutes ces méthodes font partie de la triangulation active qui sera donc présentée dans la sous-section qui suit.

2.2.1.3.a Triangulation active

Pour effectuer de la triangulation active, on se doit de créer de la lumière structurée, soit à l'aide d'un point ou d'une ligne de lumière ou d'un ensemble de franges (Blais 2001). Le projecteur et le capteur sont placés à un certain angle l'un de l'autre et l'endroit où le faisceau de lumière atteint le capteur permet de déterminer un angle qui permet à son tour de déterminer la position du point (ou ligne ou franges) dans l'espace (Besl 1989). Le capteur est une caméra ligne si l'émetteur envoie un point de lumière ou une caméra matricielle si l'émetteur envoie une ligne de lumière ou un ensemble de franges. Pour un point de lumière, l'incertitude minimale δ_z pour le laser de longueur d'onde λ est donné par :

$$\delta_z = \lambda / 2 \pi \sin \theta \sin \alpha_d \qquad (2.1)$$

Et la distance maximale pour les mesures est donné par :

$$\Delta z = 2\lambda / \sin^2 \alpha_d \qquad (2.2)$$

où sin α_d est l'ouverture du détecteur et θ est l'angle de triangulation. Bien sûr, avec un seul point de lumière, la surface $f(x, y)$ doit être balayée en x et en y.

Pour ce qui est de la projection de lignes de lumière, le balayage de la scène xy ne se fait évidemment que dans une seule dimension. Le montage de la figure 2.14 montre comment le balayage de l'objet d'intérêt a lieu. Aussi, la règle de Scheimpflug (Bigler 2002) dicte que si les plans de l'image, de détection et de l'instrumentation optique partagent tous un axe commun, le maximum de résolution pour la mesure de la distance est atteint.

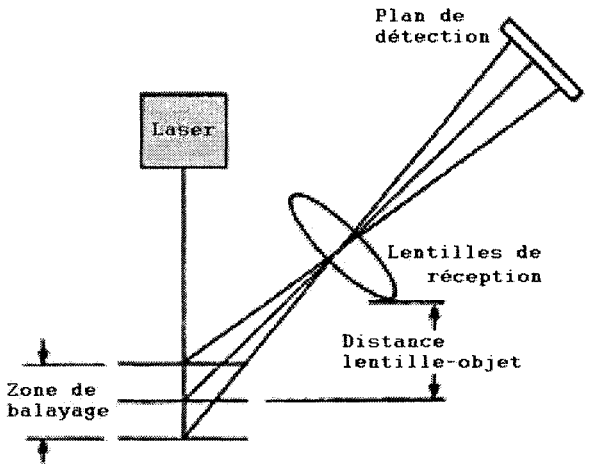


Figure 2.14: Triangulation par « lignes » de lumière

Enfin, le système illustré à la figure 2.15 (ci-contre) projette un ensemble de patrons de lumière structurée (franges) sur la scène, capture les images et calcule ensuite l'image de profondeur de la scène.

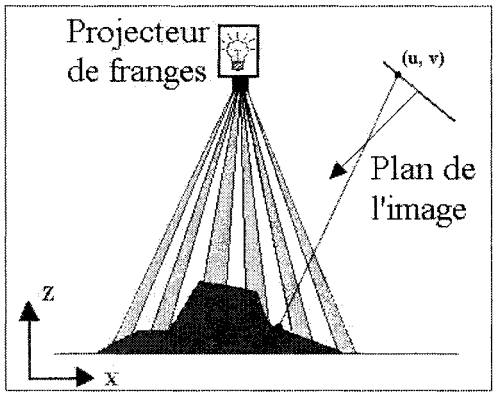


Figure 2.15: Triangulation active par franges

2.2.2 Description de la méthode InSpeck

2.2.2.1 Exemple d'acquisition

Nous avons procédé à une acquisition à l'aide d'un 3D Capturor de InSpeck. Cet appareil utilise la technique de triangulation active avec projection de franges de lumière. Sur la figure ci-contre sont présentées les différentes étapes du traitement des images lors d'une acquisition. Tout d'abord, l'image a) nous présente l'acquisition normale d'une image par la caméra CCD. Cette image nous servira plus tard de texture. À l'image b), on voit une des quatre images avec franges servant à construire l'image de phase affichée en c). Ensuite, il est important de choisir une zone d'intérêt de la région voulue de l'image et d'effectuer un déroulement de la phase pour obtenir l'image de profondeur présentée en d). Finalement, en e), on peut voir la reconstruction finale du modèle avec ou sans texture. On remarque dans le cou un petit trou vide qui se trouve dans une zone non visible par la caméra.

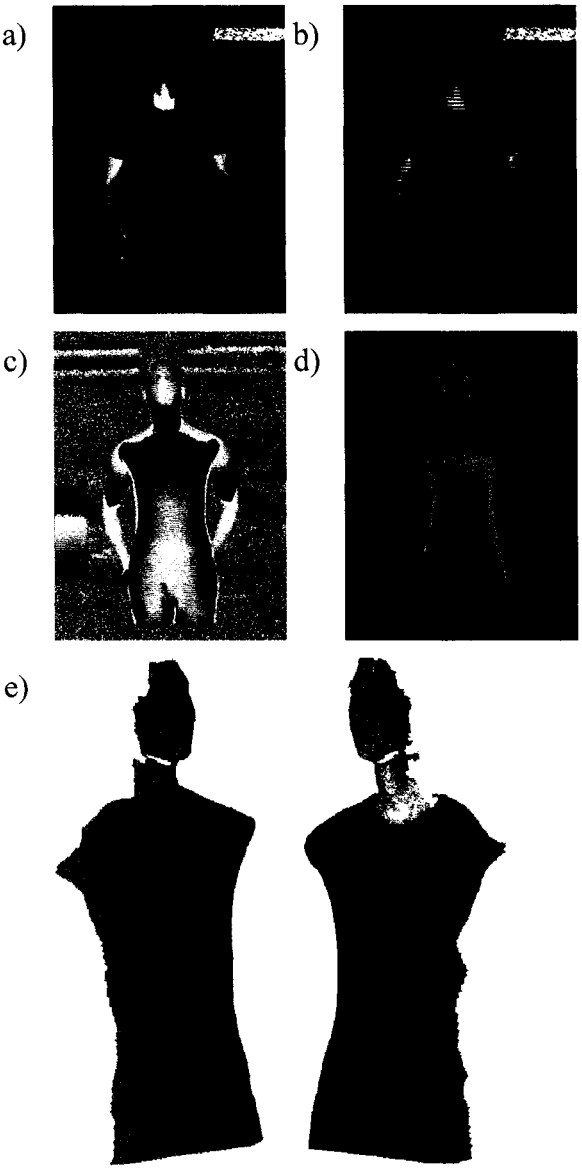


Figure 2.16: Exemple d'acquisition

2.2.2.2 Pourquoi choisir la méthode InSpeck ?

Voici une comparaison entre les spécifications de deux appareils commerciaux récents : le 3D Capturor II de InSpeck et le VIVID 910 de Minolta. Les données ont été recueillies sur les sites Internet de InSpeck et de Minolta. Le tableau suivant en présente une synthèse.

Appareil	**InSpeck 3D Capturor II**	**Minolta VIVID 910**
Technique d'acquisition	Triangulation avec projection d'un ensemble de frange lumineuse	Triangulation avec projection d'un faisceau laser
Source utilisée	Lampes halogènes	Laser de classe B
Plage numérisable	1200x900x1000 mm	1200x900x400 mm
Nombre de points possibles dans une acquisition	300 000 pts	77 000 pts (307,000 pts en mode précision)
Temps d'une d'acquisition	0.4 sec	0.3 sec (0.5 sec en mode couleur et 2.5 sec en mode précision)
Incertitude sur profondeur	0.5 mm	0.008 mm (en mode précision)
Application de texture couleur	Oui	Oui

Tableau 2.1: Comparaison de spécifications d'appareils commerciaux

Pour le 3D capturor II, la vitesse d'acquisition et l'incertitude sont indépendantes du nombre de points. Si on choisit de capturer la texture, il faut ajouter le quart du temps soit un total de 0.5 secondes pour une acquisition. Puisque l'appareil est essentiellement constitué de lampes halogènes, et d'une caméra CCD, il est relativement peu coûteux.
Les incertitudes que peut atteindre le Vivid 9100 sont beaucoup plus faibles. Par contre, ces incertitudes s'appliquent seulement au mode « précision », la compagnie ne spécifie pas quel est le niveau d'incertitude en mode rapide. Aussi, le temps d'acquisition devient très grand, ce qui augmente de beaucoup le flou de mouvement causé par le mouvement du patient durant l'acquisition et réduit la qualité de l'image. De plus, afin d'obtenir cette précision, on doit utiliser un laser et un système de balayage mécanique sophistiqué ce qui doit augmenter de beaucoup le coût de l'appareil.

2.2.3 Limites de la méthode

Maintenant que les principes de fonctionnement sont bien établis, les limites de ces méthodes peuvent être discutées. Les limites générales de la triangulation seront d'abords présentées suivi des limites spécifiques à la technologie InSpeck.

2.2.3.1 Limites de la triangulation

Tout d'abord, comme le démontre bien l'image ci-contre, un montage de triangulation active possède des limites dues au champ de vision de la caméra qui ne peut percevoir toute face cachée par une irrégularité dans la forme de l'objet à numériser.

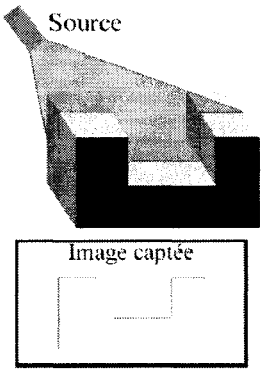


Figure 2.17: Exemple d'erreur du point de vue de la caméra

2.2.3.1.a Erreur de localisation aléatoire

Une surface dont on désire connaître la géométrie externe est soit lisse ou rugueuse, et ce, à différents degrés. Ce taux de « rugosité » est directement dû à la résolution latérale de notre système. La faiblesse de la triangulation d'un point est donc la variation dans la forme de l'image du point de lumière envoyé. Cette variation est due à ce que l'on appelle le « *speckle* », qui est en fait un phénomène optique de dispersion de la lumière qui apparaît alors sous forme de petites taches sur l'objet et qui empêche de déterminer de façon précise la position du faisceau envoyé (Jähne, Haußecker 2000). Cela donne un peu le même effet que la lumière du soleil reflétée sur un lac puisque l'image du soleil reflétée se trouve alors modifiée, à nos yeux, par la surface ondulée du lac. Ainsi, la forme observée du point de lumière envoyé dépend de la « microtopographie » de la surface observée. Ceci entraîne une erreur de localisation aléatoire dont l'écart type δz_m peut théoriquement être calculé par :

$$\delta z_m = C\lambda / 2\,\pi \sin u_{obs} \sin \theta \qquad (2.3)$$

où θ est l'angle de triangulation, sin u_{obs} est l'ouverture pour l'observation, λ est la longueur d'onde de la lumière et C est le contraste dû au *speckle*. Ce contraste est fixé à 1 pour l'utilisation du laser. Généralement, un rayon laser se distingue de la lumière ordinaire par sa directivité (ou sa divergence; faisceau visible à très grande distance), sa monochromacité (une seule couleur ou longueur d'onde), sa cohérence spatiale et temporelle (tous les photons sont dans la même phase), et sa luminance (intensité de lumière) (Bekele 2000). L'effet de *speckle* n'est pas dû à la monochromacité du laser, mais bien à sa cohérence spatiale. De plus, comme le démontre la figure ci-contre, l'erreur de localisation est à la fois inhomogène et anisotrope.

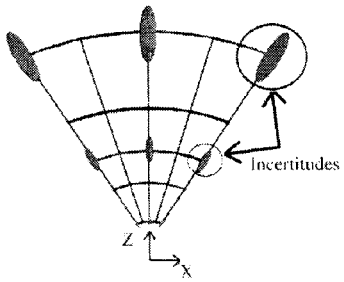


Figure 2.18: Erreur de localisation inhomogène et anisotrope

2.2.3.1.b Résolution spatiale et ombrage inhomogènes

Puisque l'illumination et la capture de la scène ne sont pas coaxiales, il est possible que certains endroits ne soient pas atteints par la lumière et que certains endroits soient hors du champ de vision du capteur. Aussi, cela entraîne que la capture d'objets peut fournir des données dont la résolution est inhomogène, ce qui peut faire en sorte que la précision de l'estimation de la profondeur n'est pas la même sur l'ensemble de l'objet.

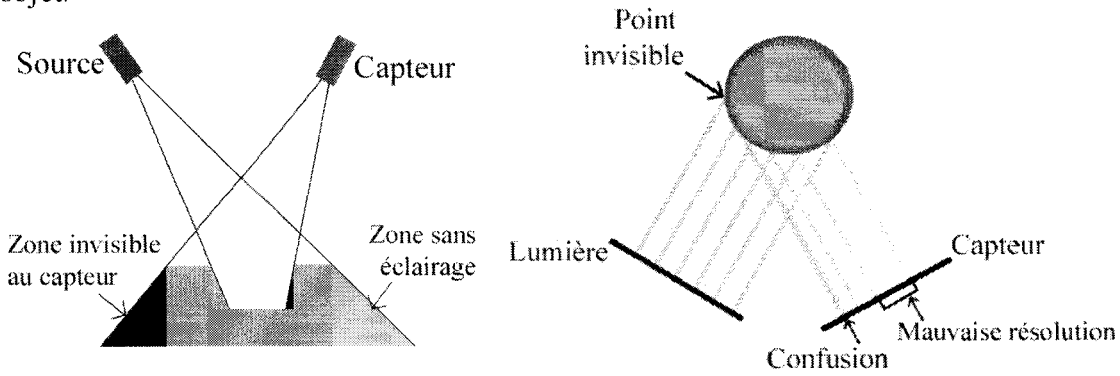


Figure 2.19: Limites de la triangulation

2.2.3.2 Limites de la méthode InSpeck

À la figure 2.8, nous avons vu qu'une des nouvelles approches de la méthode InSpeck est d'utiliser plusieurs numériseurs pour capturer une plus grande partie de la surface du tronc. Cette nouvelle approche à plusieurs numériseurs apporte toutefois plusieurs défis supplémentaires. Dans le but de pouvoir utiliser cette méthode à grande échelle, l'automatisation complète ou presque complète de toutes les parties du traitement des données se doit d'être faite.

Dans Pazos 2002, il est souligné que les opérations manuelles sont longues et répétitives et qu'un moyen doit être pris pour remédier à cette situation. Afin de pouvoir obtenir des indices cliniques valides, des algorithmes fiables et répétables doivent remplacer ces opérations manuelles. Il faut donc diminuer le nombre d'opérations manuelles nécessaires à la construction d'une représentation tridimensionnelle du tronc humain. Voici une synthèse rapide des étapes :

- Capture du tronc par les caméras (automatique);
- Génération de l'image de phase (automatique);
- Détourage de la région d'intérêt (manuel);
- Recalage des surfaces avec recouvrement (manuel);
- Construction des objets 3D (automatique).

2.2.3.2.a Problématique du détourage de la région d'intérêt

Pour l'instant, le détourage est une opération coûteuse en temps. Un opérateur doit, à l'aide de la souris, indiquer itérativement les sommets du polygone représentant la région d'intérêt. Il serait préférable de posséder un moyen rapide, efficace et reproductible pour cerner la région d'intérêt d'une image, donc du tronc humain dans le cas présent. Aussi, moyennant l'aide des gens de InSpeck Inc., il pourrait être possible de scinder toutes ces étapes afin que l'opérateur n'ait qu'à appuyer sur un bouton pour que tout s'effectue automatiquement.

Pour éliminer cette étape manuelle, il faut donc effectuer un détourage automatique de la région d'intérêt. L'automatisation de cette opération comprend en soi plusieurs étapes :

- détermination du nombre de régions dans l'image;
- association de chaque pixel à une région donnée (segmentation de l'image);
- choix des régions d'intérêt;
- détection du contour de la région (ajustement et adoucissement).

Dans chacun de ces domaines, de nouvelles études apportent sans cesse de nouvelles solutions à ces problèmes. Il serait donc nécessaire d'agencer efficacement quatre méthodes correspondant aux quatre étapes mentionnées ici. La revue de la littérature de ces méthodes sera l'objet de la section 2.3.

2.2.3.2.b Problématique du recalage

Il a été mentionné que la construction des objets 3D par les caméras est automatique, sauf pour le recalage. Le recalage sert à « re-souder » les différentes surfaces du tronc capturées à l'aide des différentes caméras. Il existe déjà un moyen d'effectuer le recalage, moyennant tout d'abord un recalage à l'aide d'une toile placée dans le domaine numérisable. Cette toile sert de référence à tous les numériseurs et le logiciel calcule alors les matrices de transformations qui relient les vues entre les diverses caméras. Afin d'éliminer cette étape manuelle, il faudra trouver un algorithme efficace de recalage automatisé approprié pour le cas du recalage de plusieurs morceaux de tronc. Il existe des algorithmes permettant un recalage rigide (sans déformation) des surfaces et d'autres qui permettent un recalage élastique (avec déformation) des surfaces.

L'acquisition de deux ou plusieurs images s'effectue à l'aide d'un montage semblable à celui de la figure 2.20. Chaque caméra voit une portion différente du sujet, ce pourquoi les surfaces à recaler ne seront que partiellement correspondantes.

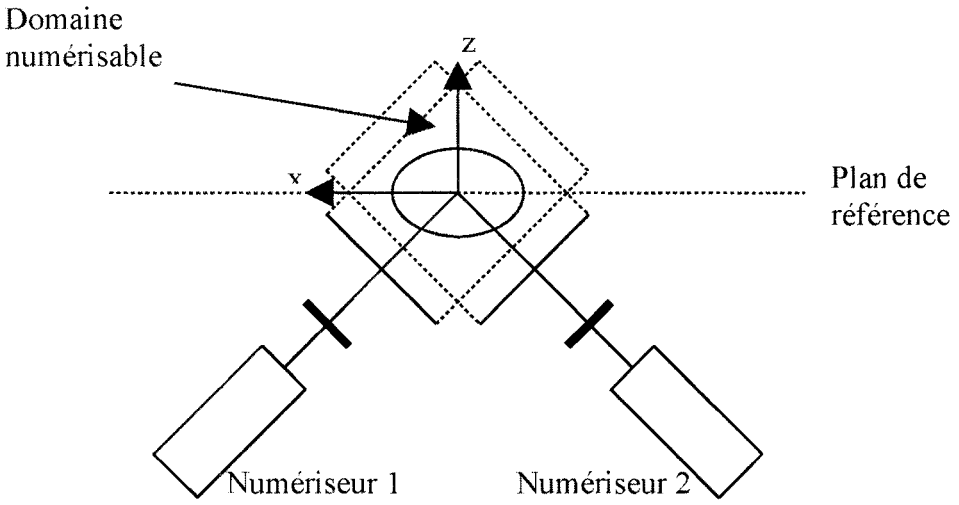


Figure 2.20: Exemple de configuration de numériseurs

Afin d'obtenir des images, le logiciel FAPS 6.0 de InSpeck inc. fournit des images de toutes les vues désirées. Ces images sont ensuite exportées dans le logiciel EM de InSpeck inc. afin d'en calculer les coordonnées tridimensionnelles et afin d'ultimement pouvoir recréer toute la surface externe voulue de l'objet. Les différentes vues doivent donc être recalées ensemble afin de former une seule surface cohérente.

Pour l'instant, l'utilisation de la toile de référence est nécessaire avant l'acquisition de chaque surface et les paramètres obtenus par ce processus sont ensuite utilisés pour effectuer le recalage. Si on revient au montage illustré à figure 2.8, on peut voir que le positionnement de la toile dans ce montage pourrait poser problème. En effet, la toile est placée de façon à être centrée et perpendiculaire aux numériseurs 1 et 2. Ceci implique que la toile n'est pas perpendiculaire aux numériseurs 3 et 4 et, dépendamment de l'angle entre ces numériseurs et le numériseur 1, il est possible que la toile ne soit visible que d'un angle relativement mince, ce qui rend le recalage moins précis. De plus, la toile est souple et est victime des courants d'air, ce qui ajoute de l'erreur.

Quoique les résultats de ce recalage ne soit potentiellement pas parfaits, ces résultats pourront tout de même servir de référence pour les comparaisons avec les

résultats obtenus à l'aide des divers algorithmes. Le but est donc de recaler des images de topographie de surface en n'utilisant, au plus, qu'une simple approximation de départ.

2.3 Techniques de détection automatique de régions d'intérêt

Afin de détecter la région d'intérêt d'une image, il faut d'abord découper l'image en régions, ce qui veut dire que l'on doit procéder à une segmentation de l'image.

Il existe deux grandes familles d'algorithmes de segmentation d'image :

- La segmentation à base de modèle (*ad hoc*) : on possède des connaissances *a priori* sur l'apparence de la région qu'on veut isoler (couleur, forme, dimension, etc.) Exemple : repérer une tumeur sur une image SPECT.
- La segmentation sans *a priori* (générale) : on ne possède aucune connaissance *a priori* sur les objets à segmenter. Exemple : séparer une photographie quelconque en différentes régions correspondant aux différents objets qui apparaissent sur l'image.

Une caractéristique importante d'un algorithme de segmentation est le niveau d'interaction requis. En somme, il y a deux grandes catégories de niveaux d'interaction :

- Les algorithmes autonomes : aucune intervention de l'utilisateur est requise.
- Les algorithmes assistés : l'utilisateur doit identifier manuellement la localisation d'une région d'intérêt. L'algorithme de segmentation se charge ensuite de calculer (ou de raffiner) les bordures de cette région.

Une fois la segmentation effectuée, il ne restera qu'à choisir quel(s) segment(s) représente(nt) le mieux la région d'intérêt. L'élaboration de critères permettant le choix de ces régions étant une étape très étroitement liée à la nature du problème, il est certain que ces critères ne seront utilisables que pour la segmentation d'images de topographie de surface. Pour cette raison, cette étape ne sera décrite en détail qu'au chapitre suivant intitulé « Méthodologie ».

2.3.1 Techniques de segmentation sans *a priori*

Pour séparer l'image en régions distinctes, sans utiliser de connaissances *a priori*, les critères de segmentation qu'on peut utiliser sont les suivants :

- Chaque région tend à avoir une certaine homogénéité d'apparence.
- L'« apparence » est différente d'une région à une autre. Donc il y a des discontinuités d'apparence aux frontières entre les régions.

Sachant cela, les sections suivantes présentent en détail la segmentation d'une image ainsi que le choix du nombre de régions dans une image donnée.

2.3.1.1 Segmentation au sens du maximum de vraisemblance

Soit une matrice X contenant les données réelles que l'on désire mesurer. Chaque élément X_i de X possède un état donné, $X_i \in \{1 \ldots K\}$, K étant le nombre d'états possibles. X ne peut pas être observé directement. Soit Y une observation de X par l'intermédiaire d'un instrument de mesure (*e.g.* une caméra). Une valeur Y_i observée suit une distribution qui dépend uniquement de la valeur X_i sous-jacente. Chaque classe possède une distribution qui lui est propre. Selon Stanford et Raftery 2002, le processus d'observation peut donc être décrit par une fonction de densité de probabilité :

$$f(Y_i | X_i = m) \tag{2.4}$$

Selon ce modèle, la réalisation d'un pixel sur une image ne dépend que de la classe du point observé et de la distribution associée à cette classe.

La détermination par maximum de vraisemblance de la classe *m* à laquelle appartient un pixel Y_i observé, est donnée par l'équation suivante :

$$\hat{X}_{MV,i} = \arg\max_m f(Y_i | X_i = m) \tag{2.5}$$

Cet estimateur peut être résolu à condition que les distributions associées à chaque classe soient connues. En pratique, il y a deux approches possibles pour modéliser la distribution associée à une classe :

- Paramétrique : Les FDPs (fonctions de densité de probabilité) suivent des formes connues qu'on peut représenter à l'aide d'une équation symbolique paramétrisée.
- Non-paramétrique : Les FDPs sont approximées au moyen de la technique des fenêtres de Parzen.

En général, l'approche non-paramétrique est plus précise parce qu'elle s'adapte spécifiquement à la nature des données des différentes classes. Cependant, l'approche paramétrique est beaucoup plus simple à réaliser et moins gourmande en temps de calcul.

2.3.1.2 Segmentation maximum *a posteriori*

Le problème d'estimation maximum *a posteriori* (MAP) de X, selon une approche paramétrique, consiste à maximiser la probabilité $P(X|Y,\Theta)$ où Θ est l'ensemble des paramètres décrivant les différentes classes. Il est à noter que dans ce contexte, la fonction à maximiser est une probabilité, plutôt qu'une densité de probabilité, parce que X est discret.

Dans un cadre Bayésien, le maximum de probabilité *a posteriori* est lié à la vraisemblance de la manière suivante décrite dans Stanford et Raftery 2002 :

$$\begin{aligned} P(X|Y,\theta) &= f(Y|X,\Theta)P(X) \\ P(X|Y,\theta) &= \prod_i f(Y_i|X_i,\Theta)P(X_i) \end{aligned} \tag{2.6}$$

Donc :

$$X_{MAP} = \arg\max_X \prod_i f(Y_i|X_i,\Theta)P(X_i) \tag{2.7}$$

Le chaînon manquant dans la formulation de la segmentation MAP est la définition de $P(X_i)$. Cette probabilité sera définie de façon à favoriser l'homogénéité des différentes régions de l'image par l'intermédiaire du modèle de Potts (Potts, 1952). Le modèle de Potts exprime la tendance qu'a chaque pixel à ressembler à ses voisins. Chaque pixel doit obligatoirement appartenir à une et seulement une des K classes possibles. Donc, la probabilité qu'un pixel X_i appartienne à la classe m dépend du nombre de voisins de X_i qui sont de la classe m. Le modèle est défini comme suit :

$$P(X_i = m) = \frac{\exp(\phi V(X_i, m))}{\sum_{\ell=1}^{K} \exp(\phi V(X_i, \ell))} \tag{2.8}$$

où la fonction $V(X_i, m)$ retourne le nombre de voisins de X_i ayant la valeur m et K est le nombre de classes, donc le nombre de valeurs possibles de m. Le paramètre ϕ indique la « force » de la tendance d'homogénéité imposée par le modèle de Potts. Le voisinage de X_i est défini comme étant l'ensemble des huit pixels directement ou diagonalement adjacents à X_i. Dans la littérature, ce type de modèle probabiliste, combinant les notions de vraisemblance et de relation de voisinage, est appelé *champ aléatoire de Markov*.

Une des grandes différences entre l'estimateur MAP et l'estimateur MV est qu'avec l'estimateur MV, tous les pixels sont considérés indépendants, ce qui rendait le problème facile à résoudre. La difficulté avec l'estimateur MAP et le modèle de Potts est qu'il faut que le voisinage de chaque pixel soit connu *a priori*, ce qui n'est pas toujours possible. Afin de pallier au problème, l'algorithme *Iterative conditional modes* (ICM) fut proposé par Besag (1986). Cet algorithme emploie une stratégie itérative pour résoudre l'estimateur MAP. L'idée fondamentale est d'utiliser les valeurs estimées à l'itération précédente dans l'évaluation de la fonction de probabilité du modèle de Potts, ce qui donne un estimateur appelé *maximum marginal a posteriori* (MMAP).

$$\hat{X}_{MMAP}^{(n)} = \arg\max_{X} \prod_i f(Y_i | X_i, \Theta) P(X_i | \hat{X}^{(n-1)}) \tag{2.9}$$

Le modèle de Potts récurrent utilisé par l'estimateur MMAP s'exprime comme suit :

$$P\left(X_i = m \middle| \hat{X}^{(n-1)}\right) \approx \frac{e^{\phi V\left(\hat{X}_i^{(n-1)}, m\right)}}{\sum_{l=1}^{K} e^{\phi V\left(\hat{X}_i^{(n-1)}, l\right)}} \tag{2.10}$$

L'algorithme ICM de base consiste à ré-appliquer l'estimateur MMAP itérativement jusqu'à ce que le résultat se stabilise. Pour que l'algorithme ICM puisse converger vers la bonne solution, il faut lui fournir une estimation initiale. Dans la méthode proposée par Stanford et Raftery (2002), l'algorithme « Marginal mixture EM segmentation » est employé pour produire une très bonne approximation initiale.

2.3.1.3 L'estimation des paramètres des classes

Jusqu'à maintenant, il a toujours été assumé que Θ était connu. En pratique, ceci est rarement le cas. Une extension à l'algorithme ICM, proposée par Besag (1986), consiste à ré-estimer Θ à chaque itération de l'ICM, en posant l'hypothèse d'ergodisme. Dans le cas de distributions gaussiennes multivariées, les estimateurs des paramètres sont les suivants :

$$\hat{\mu}_m^{(n)} = \frac{1}{\Omega\left\{Z_m^{(n)}\right\}} \sum_{i \in Z_m^{(n)}} Y_i$$

$$\hat{\Sigma}_m^{(n)} = \frac{1}{\Omega\left\{Z_m^{(n)}\right\}} \sum_{i \in Z_m^{(n)}} \left(Y_i - \hat{\mu}_m^{(n)}\right)\left(Y_i - \hat{\mu}_m^{(n)}\right)^T$$

$$Z_m^{(n)} = \left\{i \in \{1, \ldots, N\} \middle| \hat{X}_i^{(n)} = m\right\}$$

Note : Ω représente l'opérateur de cardinalité (nombre d'éléments de l'ensemble).

La variable intermédiaire $Z_m^{(n)}$ représente l'ensemble des indices des pixels qui sont assignés à la classe m à l'itération n. N représente le nombre total de pixels dans l'image.

2.3.1.4 L'estimation du nombre de classes

Jusqu'à maintenant, il a toujours été considéré que K, le nombre de classes, était connu. Dans le cas où on ne possède aucune information *a priori*, K n'est pas connu. Pour résoudre ce problème, il est possible d'utiliser le « Pseudo-likelihood information criterion » (PLIC). Le PLIC a initialement été utilisé pour segmenter des images de PET Scan (Tomographie d'Émission par Positrons (TEP)) de poumons de chien, donc dans un contexte de segmentation d'images médicales. Voici un exemple de résultats fourni par les auteurs (Stanford et Raftery, 2002) :

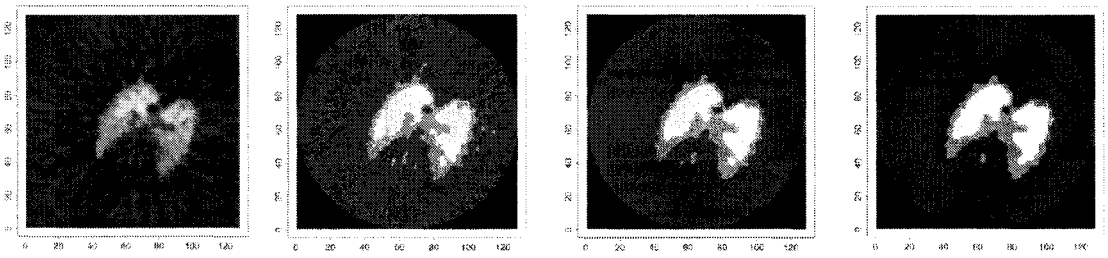


Figure 2.21: Utilisation du PLIC sur un PET Scan

Sur la figure 2.21, on peut voir les différentes étapes de la segmentation automatique de l'image. La première image à partir de la gauche est l'image initiale des poumons de chien. À partir de cette image, l'algorithme du PLIC a déterminé que le nombre idéal de segments pour cette image est de 4. La seconde image montre la segmentation initiale de l'image de poumons de chien en quatre segments. La troisième image montre la segmentation de l'image de poumons de chien en quatre segments par l'algorithme ICM initialisé par les résultats de la segmentation initiale. La quatrième et dernière image montre le raffinement du résultat par les opérations morphologiques d'ouverture et de fermeture qui seront discutées plus tard dans la méthodologie.

L'idée de base du PLIC est de trouver un compromis entre le nombre de classes et la vraisemblance de l'estimation de X. La fonction PLIC proposée par Stanford et Raftery (2002) est la suivante :

$$PLIC(K) = 2\log\left(L_{\hat{X}}(Y|K)\right) - d_K \log(N) \tag{2.11}$$

où $L_{\hat{X}}(Y|K)$ représente la pseudo-vraisemblance de la segmentation effectuée avec K classes sur $\hat{X}$, l'estimation de la scène réelle X obtenue par l'algorithme ICM. Aussi, d_K représente le nombre de paramètres utilisés pour représenter les classes et N est le nombre de pixels de l'image. Les paramètres peuvent être, par exemple, les vecteurs moyenne et les matrices de covariance des classes (modèle gaussien multivarié). Ces deux éléments sont combinés pour exprimer la complexité de l'ensemble des classes qui vient pénaliser la pseudo-vraisemblance.

Ce critère vient du « bayesian information criterion » (BIC) qui, selon la notation utilisée ici, s'exprime comme ceci :

$$BIC = 2\log\left(p\left(Y|\hat{\Theta}, C_K\right)\right) - d_K \log(N) \tag{2.12}$$

où C_K et $\hat{\Theta}$ représentent respectivement l'ensemble des K classes et l'estimateur au sens du maximum de vraisemblance des paramètres qui représentent les K classes (sous l'hypothèse d'ergodisme).

Idéalement, il serait possible de calculer la vraisemblance de l'image qui s'exprime habituellement par :

$$L(Y|K) = \sum_x p(Y|X = x, K)p(X = x|K) \tag{2.13}$$

La raison pour laquelle le BIC ne peut être utilisé intégralement est que le calcul de vraisemblance implique le calcul de toutes les possibilités de configuration d'une image. Par exemple, pour 256 couleurs sur une image de 640 par 480, cela donne un chiffre de l'ordre de 10e739811 configurations, ce qui est évidemment inacceptable.

Pour pallier à ce problème, il faut procéder au calcul de la pseudo-vraisemblance mentionnée précédemment. Plutôt que de calculer toutes les possibilités, l'idée de la pseudo-vraisemblance est d'effectuer la somme seulement sur les configurations près de l'estimation ICM de X, notée par $\hat{X}$. On considère donc chaque pixel Y_i selon X_{-i} qui symbolise X moins la valeur X_i. On en obtient une vraisemblance conditionnelle, dans laquelle $V(\hat{X}_i)$ symbolise le voisinage de $\hat{X}_i$:

$$L(Y_i|\hat{X}_{-i},K)=\sum_{j=1}^{K} p(Y_i|X_i=j)p(X_i=j_i|V(\hat{X}_i)) \qquad (2.14)$$

où $p(Y_i|X_i=j)$ ne requiert que l'évaluation de la densité conditionnelle de Y_i selon X_i et où $p(X_i=j_i|N(\hat{X}_i))$ est évaluée par la seconde formule de la section « modèle de Potts ».

Les vraisemblances conditionnelles sont combinées pour former la pseudo-vraisemblance de l'image qui, par une maximisation de $p(y_i|x_i,\sigma^2)$ selon Qian-Titterington 1993, peut s'exprimer par :

$$L_{\hat{X}}(Y|K)=\prod_i f(Y_i|\hat{X}_{-i},\hat{\phi})=\prod_i f(Y_i|X_i=j)p(X_i=j|V(\hat{X}_i),\hat{\phi}) \qquad (2.15)$$

Maintenant, si on revient à la formule du BIC, on peut remplacer la vraisemblance $L(Y|X)$ qui demande beaucoup trop de calculs par la pseudo-vraisemblance $L_{\hat{X}}(Y|K)$ facile à calculer pour obtenir la nouvelle approximation :

$$PLIC(K)=2\log(L_{\hat{X}}(Y|K))-d_K\log(N) \qquad (2.16)$$

Finalement, selon Stanford et Raftery 2002, afin de maximiser $PLIC(K)$, on commence par effectuer le calcul avec $K = 2$, on incrémente la valeur de K de 1 à chaque itération et on choisit le premier maximum local de K comme le choix pour le nombre de segments K de l'image.

2.4 Recalage automatique des surfaces

Avec la méthode InSpeck, plusieurs numériseurs doivent être utilisés afin d'obtenir toutes les surfaces nécessaires pour représenter la surface avant du tronc. Une fois que ces surfaces sont obtenues, il faut les fusionner, c'est-à-dire qu'il faut procéder au recalage des surfaces.

Dans les sections qui suivent, une revue de la littérature des différentes méthodes de recalage sera présentée. Dans ce mémoire, l'intérêt est porté sur le recalage de surface 3D à surface 3D. Il existe aussi plusieurs méthodes spécifiquement développées pour le recalage 2D-2D et 2D-3D, mais ce sujet serait trop vaste pour être traité entièrement ici et, dans le contexte de ce projet, seul le recalage 3D-3D est utilisé, ce pourquoi seules ces techniques seront exposées de façon précise.

Le recalage est une technique permettant de déterminer quelle transformation doit être appliquée à une image afin que les points correspondants soient superposés sur une seconde image servant de référence. En sachant cela, une bonne quantité de méthodes différentes ont été développées afin de procéder au recalage de différentes images. Les méthodes décrites dans la littérature se basent sur différents types de transformations, telles les transformations rigides (rotation, translation, homothétie), affines (cisaillement), de projection (correspondance des droites sans préservation du parallélisme) ou courbes (correspondances des lignes droites avec les lignes courbes), afin que les images se superposent correctement.

Essentiellement, dans la pratique, le recalage implique souvent un problème d'appariement de points. Pour deux surfaces numérisées en 3D d'un même objet dans des angles de vue semblables, il existe potentiellement des points correspondants sur chaque surface. Le choix de ces points se fait habituellement manuellement par un opérateur ou par des algorithmes semi-assistés où l'opérateur doit fournir des indices à l'algorithme. Il existe aussi des techniques complètement automatisées, mais ces algorithmes se doivent d'être extrêmement robustes, ce qui est assez difficile à réaliser.

2.4.1 Techniques de recalage 3D-3D

Le rôle de cette section est de présenter des algorithmes de recalage permettant de recaler deux surfaces tridimensionnelles qui ne sont que partiellement correspondantes. Ces algorithmes doivent contenir aussi peu d'interventions humaines que possible.

Dû à la grande difficulté du problème et à la satisfaction relativement faible qu'ont apporté plusieurs solutions proposées dans la littérature afin de solutionner ce problème en particulier, plusieurs méthodes ont été implémentées afin d'effectuer le recalage. Bien que beaucoup de méthodes soient très utiles dans des cas très particuliers, le fait que l'on tente ici de recaler une partie indéterminée d'une surface sur une partie indéterminée d'une autre surface ajoute beaucoup d'inconnus au problème et réduit le nombre de solutions applicables.

Le recalage de deux surfaces peut être rigide (uniquement les transformations rigides) ou élastique (utilisation des transformations affines, de projection ou courbes et utilisation de modèles de déformations).

Tout d'abord, pour le recalage rigide, une première option simple pour effectuer le recalage 3D est de sélectionner des points correspondants sur les deux images de texture et de se servir de ces points pour ensuite effectuer le recalage 3D. Une autre option est de procéder directement à l'appariement des points tridimensionnels entre eux afin de tenter de replacer les images correctement l'une juxtaposée à l'autre.

Pour le recalage élastique, une méthode utilisant les splines plaque mince (Bookstein 1989) est décrite à l'annexe 2. Le recalage élastique a été étudié, mais il offre trop de degrés de liberté possibles, c'est-à-dire qu'il y a potentiellement plusieurs déformations différentes possibles pour agencer les deux surfaces. Puisque, au départ, la transformation rigide requise pour recaler les deux surfaces ensemble n'est pas connue, il est possible de déformer deux parties non-correspondantes des deux surfaces afin de les faire correspondre, ce qui est, de toute évidence, une erreur très grave.

2.4.1.1 Recalage 3D par la méthode des moindres carrés

Avant de commencer cette méthode, on suppose que la correspondance entre les points d'une image et ceux d'une autre image a été établie par l'usager et qu'il ne reste plus qu'à procéder au recalage à partir des points. Cette mise en correspondance des points dépend de l'usager et n'est donc pas reproductible. Dans la littérature, il existe plusieurs algorithmes de détection de points d'intérêt comme, par exemple, trouver des coins dans l'image grâce à une matrice Hessienne (Zitova 2000, Hladůvka 2001). Ces algorithmes fonctionnent relativement bien, mais lorsqu'arrive le temps d'effectuer l'appariement automatique des points trouvés sur deux images provenant de caméras regardant la scène de deux points de vue très différents, aucun algorithme ne s'est montré assez robuste pour effectuer des appariements sans connaissances *a priori* sans erreur. Ceci explique pourquoi on suppose pour cette première méthode que l'usager doit intervenir dans le choix des points.

Une fois que les deux ensembles de points d'intérêt sont connus et que la correspondance entre les points des deux ensembles est établie par l'usager, il ne reste plus qu'à effectuer le recalage. Considérons d'abord un modèle idéal. Ce que nous savons, c'est que, dans le cas du recalage rigide, la seconde image a subi une rotation et une translation par rapport à la première image. Notre but est de faire subir la transformation inverse à notre image. Il est à noter que la transformation trouvée par la méthode des moindres carrés comprend d'autres types de transformations comme l'homothétie. Mathématiquement, cela s'exprime ainsi :

Soit X1 et X2, les ensembles de points d'intérêt des images 1 et 2, nous savons que :

- Une rotation entraîne : $X1 = R * X2$ avec R une matrice 3x3
- Une homothétie entraîne : $X1 = H * X2$ avec $H \in \mathbb{R}^3$
- Une translation entraîne : $X1 = T + X2$ avec $T \in \mathbb{R}^3$

En coordonnées homogènes, les rotations, homothéties et translations peuvent s'exprimer sous la forme d'une matrice 4x4 $\in \mathbb{R}^4$ et être combinées dans une seule matrice. Ainsi, en supposant que X1 et X2 sont en coordonnées homogènes, en faisant subir à X2 une rotation, une homothétie puis une translation, on trouve une seule matrice de transformation formée ainsi :

RH = R*H (RH, R et H sont toutes des matrices 4x4)

En ajoutant la translation, nous obtenons une matrice de la forme :

$$\begin{pmatrix} X1_{11} & X1_{12} & X1_{13} & \dots & X1_{1N} \\ X1_{21} & X1_{22} & X1_{23} & \dots & X1_{2N} \\ X1_{31} & X1_{32} & X1_{33} & \dots & X1_{3N} \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix} = \begin{pmatrix} RH_{11} & RH_{12} & RH_{13} & T_x \\ RH_{21} & RH_{22} & RH_{23} & T_y \\ RH_{31} & RH_{32} & RH_{33} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} X2_{11} & X2_{12} & X2_{13} & \dots & X2_{1N} \\ X2_{21} & X2_{22} & X2_{23} & \dots & X2_{2N} \\ X2_{31} & X2_{32} & X2_{33} & \dots & X2_{3N} \\ 1 & 1 & 1 & \dots & 1 \end{pmatrix}$$

La matrice résultante est donc la matrice de transformation globale et sera dorénavant désignée par la lettre A pour former :

$$X1 = A * X2$$

Où $A = \begin{pmatrix} RH_{11} & RH_{12} & RH_{13} & T_x \\ RH_{21} & RH_{22} & RH_{23} & T_y \\ RH_{31} & RH_{32} & RH_{33} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Sachant que X1 = A * X2, nous en déduisons que :

$$X1 * X2^T = A*X2*X2^T \qquad (2.17)$$

En multipliant par $(X2 * X2^T)^{-1}$, nous obtenons :

$$X1 * X2^T * (X2 * X2^T)^{-1} = A * X2 * X2^T * (X2 * X2^T)^{-1}$$
$$= A$$

La matrice A s'exprime ainsi par une fonction de X1 et X2, les 2 ensembles de points d'intérêt en coordonnées homogènes. Nous obtenons donc que :

$$A = X1*X2^T*(X2*X2^T)^{-1} \qquad (2.18)$$

Ceci nous permet de reconnaître l'expression de l'estimateur des moindres carrés. Ainsi, il nous suffit de retrouver la matrice de transformation grâce à l'estimateur des moindres carrés pour ensuite l'appliquer à notre image 3D afin de replacer les images ensemble.

Dans le cas idéal, cet estimateur effectue un recalage parfait d'une image par rapport à l'autre. Dans le cas réel, plusieurs facteurs influencent le recalage, tels le bruit d'acquisition des caméras et les déformations élastiques (non-uniformes sur toute l'image). Cependant, dans le contexte actuel, aucune connaissance *a priori* n'est disponible, ce qui nous empêche d'ajouter la matrice de covariance normalisée du bruit à notre équation. En transposant les matrices X1 et X2, nous pouvons obtenir une nouvelle matrice de transformation qui, au lieu d'être énoncée par :

$$X1 = A * X2$$

nous pouvons la reformuler comme étant :

$$X1^T = X2^T * B$$

ce qui nous fournira une matrice B différente de A, mais qui exprime néanmoins la même transformation. Aussi, selon la même logique :

$$B = (X2 * X2^T)^{-1} * X2 * X1^T$$

Et en connaissant la matrice de covariance normalisée du bruit R_B, nous pourrions écrire :

$$B = (X2 * R_B^{-1} * X2^T)^{-1} * X2 * R_B^{-1} * X1^T \quad (2.19)$$

Pour obtenir la forme standard de l'estimateur au sens du maximum de vraisemblance.

Cependant, en pratique, cette matrice n'est pas connue, ce qui fait que nous supposons un bruit d'acquisition comme étant un bruit blanc gaussien, ce qui fait que la matrice de covariance normalisée du bruit est la matrice identité I et que l'estimateur au sens du maximum de vraisemblance revient alors à l'estimateur des moindres carrés.

2.4.1.2 Deuxième méthode : Iterative Point Matching (IPM)

Puisque la première méthode n'a fourni des résultats satisfaisants que lorsque les points étaient sélectionnés manuellement, une seconde méthode a dû être recherchée, puis implantée, afin de tenter d'obtenir de meilleurs résultats. Cette méthode est essentiellement constituée d'un algorithme itératif qui calcule les distances séparant les surfaces 3D et qui tente de les rapprocher jusqu'à ce qu'elles soient l'une vis-à-vis l'autre (voir Zhang 1994 et Granger 2002). Zhang 1994 propose l'algorithme IPM pour le recalage de courbes 2D et Granger 2002 utilise cet algorithme pour recaler diverses données 3D d'os de mâchoires inférieures humaines *ex vivo*.

En gros, l'algorithme trouve les points d'une surface les plus près de l'autre surface, choisit un sous-ensemble de points selon un critère de distance maximale D qui sera plus longuement élaboré lors de la discussion, puis calcule le mouvement à effectuer et recommence toutes ces étapes à l'itération suivante. L'établissement du critère D permettant un bon choix de sous-ensembles de points et le choix d'un critère d'arrêt sont les deux paramètres modifiant la qualité du recalage final. Les prochaines sections présentent le fonctionnement de cette méthode résumée ici dans l'algorithme suivant.

```
TANT QUE (La convergence du mouvement requis n'est pas atteinte)
   Trouver les points de la surface de référence les plus proches de
      ceux de la surface à l'étude
   Calculer le mouvement requis pour juxtaposer les surfaces
   Appliquer le mouvement sur la surface à l'étude
ITÉRATION SUIVANTE
```

2.4.1.2.a Recherche des distances minimales

Lors de chacune des itérations de l'algorithme, il est nécessaire d'apparier chacun des points de la première surface au point le plus proche se trouvant sur la deuxième surface. Pour effectuer cette opération on peut balayer chacun des points de la deuxième surface pour chacun des points de la première surface pour trouver le point le plus proche (voir l'algorithme ci-dessous).

X est l'ensemble de points de la première surface
Y est l'ensemble des points de la deuxième surface
CP l'ensemble de points les plus proches de chacun des points de la première surface

```
Pour chaque i | xi ε X
   cpi=y0
   Pour chaque j | yj ε Y
      Si  ||xi-yj|| < || cpi ||
          cpi  = yj
retourner cp
```

Cependant, si la première surface comporte *n* points et que la deuxième surface comporte *m* points, alors cet algorithme est de complexité *O(nm)*. La conséquence de cette complexité algorithmique élevée est une perte de performance importante lorsque le nombre de points croît. Dans le cas présent, cet algorithme s'exécutait en approximativement 10 minutes (pour environ 15 000 points sur chaque surface, exécuté sur un Celeron 1.2 GHz). Ce temps d'exécution est inacceptable si l'on prend en compte que l'algorithme IPM est un algorithme itératif et qu'il n'est pas rare que plus de 30 itérations soient nécessaires pour qu'il y ait convergence.

Ainsi, il a été nécessaire de recourir à un algorithme de recherche géométrique en utilisant un arbre binaire afin d'accélérer l'opération de recherche des points de distances minimales. L'idée de base est de diviser récursivement l'espace dans lequel est contenu la deuxième surface et d'ainsi créer un arbre binaire. Par la suite, on effectue des recherches dans l'arbre binaire afin de trouver le point de la deuxième surface se trouvant le plus près d'un point donné de la première surface (voir algorithme ci-dessous). Le lecteur intéressé par le type d'arbre particulier qui a été employé peut se référer à Bonet et Peraire 1991.

```
TrouverPointsPlusProche(X,Y)

arbre = CreerArbre(Y,0,findBoundingBox(Y))
Pour chaque i | xi ε X
   cpi=RecherchePlusProche(arbre,0,y0,xi)
retourner cp
```

```
CreerArbre(Y, niveau, boundingBox)

Si niveau modulo 3 == 0
   blow, bhigh = separerEnX(boundingBox)
Si niveau modulo 3 == 1
   blow, bhigh = separerEnY(boundingBox)
Si niveau modulo 3 == 2
   blow, bhigh = separerEnZ(boundingBox)

X = (y_i ε Y | ( i>0 & y_i dans blow ))
Y = (y_i ε Y | ( i>0 & y_i dans bhigh ))

arbre.boundingBox = boundingBox
arbre.point= y_i
arbre.low  = null
arbre.high = null
Si X ≠ ∅
   arbre.low  = CreerArbre(X,niveau+1,blow)
Si Y ≠ ∅
   arbre.high = CreerArbre(Y,niveau+1,bhigh)
retourner arbre
```

```
RecherchePlusProche(arbre, niveau, ptBest, ref)

Si || ref - arbre.point || < || ref - ptBest ||
   ptBest = arbre.point

Si ref est dans arbre.low.boundingBox
   premier = arbre.low
   second  = arbre.high
Sinon
   premier = arbre.high
   second = arbre.low

Si distanceMin(premier.boundingBox, ref) < || ref - ptBest ||
   ptlow  = RecherchePlusProche(premier,niveau+1,ptBest,ref)
   Si || ref - ptlow || < || ref - ptBest ||
      ptBest = ptlow
Si distanceMin(second.boundingBox, ref) < || ref - ptBest||
   pthigh = RecherchePlusProche(second,niveau+1,
   Si || ref - pthigh || < || ref - ptBest ||
      ptBest = pthigh
retourner ptBest
```

En moyenne cet algorithme s'effectue en $O(n \log m)$. Cependant, pour des cas particuliers l'algorithme peut dégénérer en $O(nm)$. Dans ce cas-ci, le calcul des points de distance minimum s'effectue en un temps variant entre 1 et 20 secondes (comparativement à 10 minutes pour le premier algorithme).

Ci-dessous est présenté un exemple de création d'arbre et de recherche du point le plus proche. L'étoile rouge correspond au point pour lequel on recherche le point le plus proche. Les flèches rouges dans l'arbre à droite montre les nœuds de l'arbre qui doivent être parcouru afin de trouver le point le plus proche de l'étoile rouge. Ainsi, on remarque que seulement trois nœuds sont visités afin de trouver le point le plus proche.

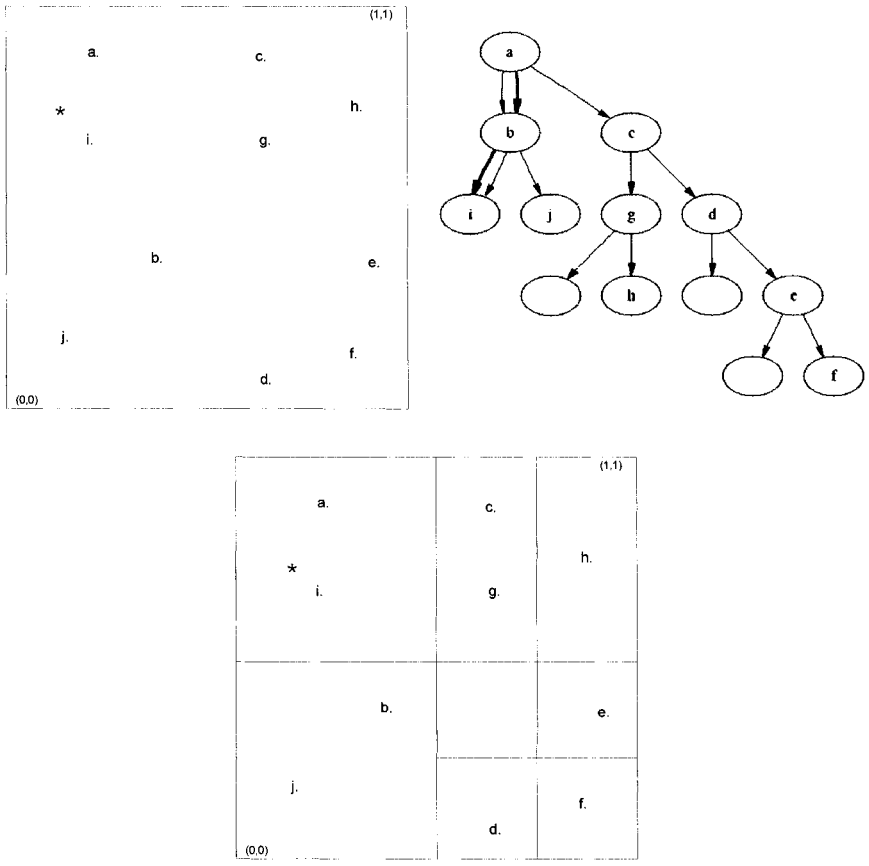


Figure 2.22: Nuage de points bidimensionnel (en haut à gauche), arbre correspondant (en haut à droite), partition récursive de l'espace obtenu à l'aide de l'arbre (en bas)

2.4.1.2.b Élimination de correspondances

Après avoir trouvé le point de la seconde surface le plus proche de chacun des points de la première surface, il est nécessaire d'éliminer un certain nombre de ces correspondances potentiellement erronées. Plusieurs critères peuvent être utilisés pour faire ce travail.

Besl et McKay 1992 supposent que les deux surfaces correspondent complètement et n'utilisent aucun critère permettant d'éliminer des correspondances. Zhang 1994 détermine un critère de distance maximale grâce à la distribution des distances (en utilisant la moyenne et l'écart type). Il a été possible de constater que ce type de critère est efficace si la surface à recaler est une partie de la surface référence. Ce type de critère ne fonctionne donc pas bien lorsque les deux surfaces correspondent partiellement.

Dans le présent cas, les deux surfaces ne correspondent que partiellement, alors les critères utilisés dans la littérature sont peu performants. Ainsi, il a été choisi d'éliminer une proportion constante des correspondances fournies par l'algorithme des distances minimales. La classification des points les plus proches s'effectue par ordre de distance et il n'est conservé qu'une certaine proportion des points ayant la distance minimale la plus faible (la proportion choisie est la proportion approximative de chevauchement des surfaces).

2.4.1.2.c Déplacement de l'image trouvée

Encore une fois, la notion de moindres carrés revient, mais dans une toute autre optique. Ici, on recherche une rotation et une translation permettant de déplacer la première surface le plus près possible de la surface de référence en minimisant la distance au sens des moindres carrés. Puisque les points utilisés pour effectuer ce calcul sont les points les plus près et non les points correspondants, un algorithme très différent doit être utilisé.

Soit X l'ensemble de points d'un maillage (ensemble de coordonnées 3D) et Y l'ensemble des points du second maillage les plus près de chacun des points de X et soit x_i un point de X et y_i le point de Y correspondant à x_i , notre but est de minimiser la distance séparant tous les x_i des y_i en minimisant la fonction de moindres carrés suivante:

$$F(R,t) = \frac{1}{N}\sum_{i=1}^{N}\left\|Rx_i + t - y_i\right\|^2 \qquad (2.20)$$

Afin de minimiser cette fonction, plusieurs méthodes existent. Dans Lorusso, Eggert et Fisher 1995, on compare les quatre algorithmes les plus populaires pour calculer la transformation qui minimise la distance qui sépare deux ensembles de points. Ces quatre méthodes sont :

- La décomposition par valeur singulière (*Singular Value Decomposition*);
- Les matrices orthonormales (*Orthonormal Matrices*);
- Le quaternion unitaire (*Unit Quaternion*);
- Le quaternion dual (*Dual Quaternion*).

Les erreurs obtenues par l'utilisation de ces méthodes sont toutes suffisamment petites pour que l'on puisse les utiliser avec confiance. Le choix s'est arrêté sur la méthode du quaternion dual puisque, pour de larges ensembles de points, elle est plus rapide. Aussi, cette méthode ne permet pas l'opération de symétrie où une surface se retrouve en version miroir d'elle même, ce qui est primordial pour le recalage de surfaces. La méthode du quaternion dual sera donc décrite ici.

2.4.1.2.c.a Méthode du quaternion dual

Un quaternion peut être vu comme un vecteur à quatre dimensions $[q_1\ q_2\ q_3\ q_4]^T$ ou comme un vecteur de 3 dimension $\vec{q} = [q_1\ q_2\ q_3]^T$ combiné avec un scalaire q_4 (voir Annexe 1).

Un quaternion dual $\hat{q}$ est composé de deux quaternions q et s selon la relation :

$$\hat{q} = q + \in s \tag{2.21}$$

où, selon une règle de multiplication spéciale pour $\in$, $\in^2 = 0$. Aussi, il est important de mentionner qu'une déformation rigide est obtenue lorsque $q^T q = 1$ et $q^T s = 0$. Alors, en définissant les matrices suivantes :

$$K(\bar{q}) = \begin{bmatrix} 0 & -q3 & q2 \\ q3 & 0 & -q1 \\ -q2 & q1 & 0 \end{bmatrix} \quad Q(q) = \begin{bmatrix} q4I + K(\bar{q}) & \bar{q} \\ -\bar{q}^T & q4 \end{bmatrix} \quad W(q) = \begin{bmatrix} q4I - K(\bar{q}) & \bar{q} \\ -\bar{q}^T & q4 \end{bmatrix}$$

On en arrive à la définition de la matrice de rotation R selon :

$$R = \left(q_4{}^2 - \bar{q}^T \bar{q}\right) I + 2\bar{q}\bar{q}^T + 2q_4 K(\bar{q}) \tag{2.22}$$

et le vecteur de translation t est égal au vecteur p du quaternion $p = [P; p_4]$ dans lequel $p_4 = 0$ obtenu par :

$$p = W(q)^T s \tag{2.23}$$

En résumé, si nous prenons un point x et que nous désirons lui appliquer une transformation afin de le faire tendre vers un point y, la transformation peut s'exprimer par :

$$Rx + t = W(q)^T s + W(q)^T Q(q) x \tag{2.24}$$

Maintenant que nous connaissons théoriquement la transformation à appliquer, il ne reste plus qu'à trouver le quaternion q qui minimisera la fonction des moindres carrés désormais exprimée par :

$$F = \frac{1}{N}\left[q^T C_1 q + N s^T s + s^T C_2 q + const.\right] \tag{2.25}$$

où $C_1 = -2\sum_{i=1}^{N} Q(y_i)^T W(x_i)$, $C_2 = 2\sum_{i=1}^{N}\left[W(x_i) - Q(y_i)\right]$ et $const. = \sum_{i=1}^{N}\left(x_i{}^T x_i + y_i{}^T y_i\right)$

Ainsi, le quaternion dual $\hat{q}$ optimal s'obtient en minimisant :

$$F' = \frac{1}{N}\left[q^T C_1 q + Ns^T s + s^T C_2 q + const. + \lambda_1\left(q^T q - 1\right) + \lambda_2\left(s^T q\right)\right] \tag{2.26}$$

où λ_1 et λ_2 sont les multiplicateurs de Lagrange (voir Annexe 1). En calculant les dérivées partielles :

$$\begin{aligned} \frac{\partial F'}{\partial q} &= \frac{1}{N}\left[\left(C_1 + C_1^T\right)q + C_2^T s + 2\lambda_1 q + \lambda_2 s\right] = 0 \\ \frac{\partial F'}{\partial s} &= \frac{1}{N}\left[2Ns + C_2 q + \lambda_2 q\right] = 0 \end{aligned} \tag{2.27}$$

En multipliant la dérivée partielle de F' par rapport à s par q, on obtient que $\lambda_2 = -q^T C_2 q = 0$ car la matrice C_2 est antisymétrique et de diagonale nulle. Alors, s est donné par :

$$s = -\frac{1}{2N} C_2 q \tag{2.28}$$

En substituant s dans l'équation de la dérivée partielle de F' par rapport à q, on obtient une équation de la forme :

$$Aq = \lambda_1 q \qquad \text{où } A = \frac{1}{2}\left[\frac{1}{2N} C_2^T C_2 - C_1 - C_1^T\right].$$

Ceci fait en sorte que q est le vecteur propre de la matrice A correspondant à la valeur propre λ_1. En substituant cela dans l'équation de F', on s'aperçoit alors que :

$$F' = \frac{1}{N}\left[const. - \lambda_1\right] \tag{2.29}$$

Donc, pour minimiser le tout, il faut trouver la plus grande valeur de λ_1, ce qui fait que le quaternion q est le vecteur propre de A correspondant à la plus grande valeur propre de A. Du quaternion q, on peut en déduire la matrice de rotation et le vecteur de translation à partir des équations fournies précédemment pour enfin pouvoir procéder à la transformation de points de X et poursuivre l'itération en cours.

2.4.1.2.c.b Quaternion dual VS Moindres carrés

Puisque la méthode des moindres carrés prend comme entrée deux ensembles de points et qu'elle les utilise pour calculer la transformation, on pourrait penser que cette méthode pourrait être utilisée dans le recalage par la méthode IPM au lieu des quaternions, ce qui est faux.

Tout d'abord, la méthode des moindres carrés est faite pour calculer la transformation entre deux ensembles de points appariés et correspondants sur les deux surfaces, ce qui n'est pas le cas ici. Ensuite, contrairement à la méthode du quaternion dual qui ne permet de calculer que les mouvements de rotation et de translation entre les deux ensembles de points, la méthode des moindres carrés permet aussi de calculer d'autres types de transformation comme des homothéties. Cela pourrait sembler désavantageux de la part de la méthode du quaternion dual, mais il est bon de ne pas détecter les homothéties dans ce cas-ci, car on ne veut pas redimensionner la surface à chaque itération.

En effet, dans le contexte de la méthode InSpeck, les acquisitions des deux surfaces sont effectuées par deux numériseurs InSpeck semblables, ce qui fait que le facteur d'échelle est le même. Puisqu'il n'existe qu'une transformation rigide (translation et rotation) entre les deux surfaces et qu'une des hypothèses nécessaires à l'utilisation la méthode IPM est que le déplacement entre les deux surfaces est petit ou connu approximativement, il est possible de considérer que les deux surfaces sont dans la même échelle.

Aussi, lorsque les deux surfaces sont relativement éloignées, il est possible d'atteindre le cas limite où tous les points d'une surface sont tous associés au même point de l'autre surface. Cette situation est montrée par un exemple sur la figure 2.23.

Un redimensionnement de la surface (homothétie) lors de la méthode IPM ferait en sorte que tous les points d'une surface convergent vers le même point et disparaissent ainsi dans un espace infinitésimal.

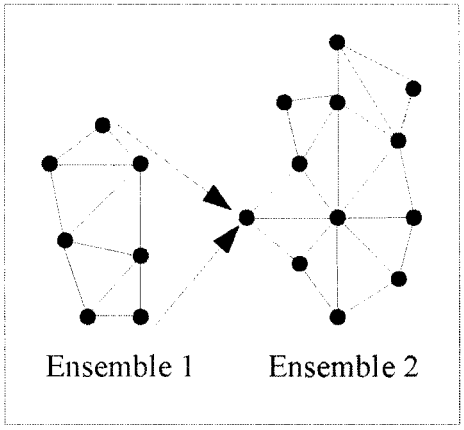


Figure 2.23: Convergence d'une surface vers un espace infinitésimal

Il n'est cependant pas recommandé d'avoir autant de distance entre les deux surfaces puisque l'utilisation de la méthode IPM implique que l'approximation de départ se doit d'être relativement près de la réalité.

2.4.1.3 Point fingerprints

La méthode des *point fingerprints* (Sun 2003) est une méthode d'appariement automatique des points correspondants de deux surfaces. Cet algorithme est utilisé pour pallier au problème principal lié à l'utilisation de la méthode IPM qui est de réussir à fournir une approximation initiale suffisamment bonne à l'algorithme IPM pour que celui-ci fonctionne correctement. La méthode des *point fingerprints* (PF) sera donc présentée et ensuite détaillée dans les pages suivantes.

Dans Sun 2003, on utilise la méthode PF pour recaler deux images acquises de la même scène vue de deux angles différents, ce que l'on cherche à faire. Ils ont utilisés des visages humains pour valider leur méthode. La figure 2.24 présente un exemple de résultat.

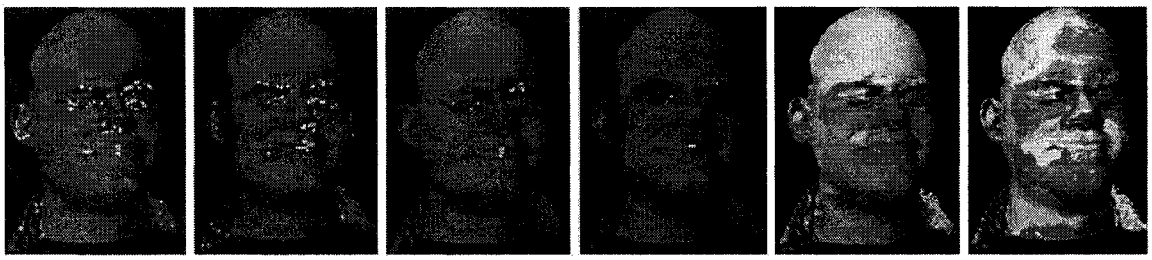


Figure 2.24: Utilisation de la méthode des *point fingerprints* sur un visage humain

En partant de la gauche, les deux premières images montrent les points d'intérêt trouvés par la méthode PF sur chacune des deux surfaces. Ensuite, les deux images du centre montrent le résultat de l'appariement des points d'intérêts. La cinquième image est le résultat du recalage et la dernière image est le raffinement par IPM.

Comme on le verra dans cette section, la méthode PF se sert des déformations locales pour trouver des points d'intérêt et, puisqu'un visage humain comporte beaucoup de déformations locales, la méthode fonctionne très bien ici. Le principe général de la méthode est de calculer pour chaque point de chaque surface un ensemble de courbes permettant de discriminer les points les uns des autres. Ces courbes sont en fait un ensemble de points qui se situent tous à distance égale du point de départ. La distance utilisée pour les calculs est la distance géodésique, c'est-à-dire la distance parcourue entre deux points en suivant la surface, donc en tenant compte des variations dans le relief. Par exemple, sur une surface plane, une courbe située à une distance géodésique unitaire du point de départ serait un cercle de rayon unitaire. Plus la surface se déforme, plus la courbe se déforme elle aussi afin d'épouser la surface. Une fois qu'un ensemble de courbes est acquis, ces courbes sont projetées sur le plan possédant la même normale que le point de départ. Ceci fournit un ensemble de courbes 2D permettant de discriminer les points entre eux. Les courbes les plus déformées se situent dans les endroits contenant le plus de déformations locales. L'appariement des courbes permet alors d'obtenir un appariement des points, ce qui fournit une première approximation de recalage en utilisant ces points dans une méthode comme le quaternion dual.

2.4.1.3.a Algorithme : Point fingerprints

```
POUR chaque surface

      POUR chaque point
            - Calculer les distances géodésiques entre le point courant
              et ses voisins situés à moins d'une certaine distance
            - Interpoler les points à plusieurs distances géodésiques
              prédéterminées plus petites que la distance maximale
            - Projeter ces points interpolés sur un plan dont la
              normale est la normale du point courant
            - Tracer un ensemble de courbes à partir des points
              possédant les mêmes distances géodésiques (point
              fingerprint)

      - Choisir les courbes (point fingerprints) présentant le meilleur
        rapport entre le rayon minimal et le rayon maximal

- Apparier les courbes grâce à une corrélation
- Utiliser une méthode (comme par exemple celle du quaternion dual)
  pour recaler les surfaces grâce aux points trouvés
- Raffiner le recalage avec IPM
```

2.4.1.3.b Calcul des distances géodésiques

Le calcul correct des distances géodésiques est l'élément clé qui fait le succès de l'algorithme. Puisque la méthode PF se doit d'être indépendante de l'échantillonnage et de la triangulation de la surface, il est important d'obtenir la distance géodésique réelle qui existe entre deux points et non pas les distances de Manhattan. Les distances de Manhattan peuvent être obtenues par l'algorithme très connu du plus court chemin de Dijkstra. Pour mieux comprendre, voici un exemple.

Supposons qu'une personne se trouve sur le coin d'un parc rectangulaire longé de tous les côtés par un sentier (Figure 2.25). Si la personne désire se rendre au coin opposé, elle peut passer par le chemin le plus court qui consiste à longer la bordure du parc (ligne continue) ou ne pas emprunter de chemin et traverser le parc en diagonal (ligne pointillée).

Dans le premier cas, la personne parcours l'équivalent d'une distance de Manhattan et dans le second cas, elle parcourt l'équivalent d'une distance géodésique.

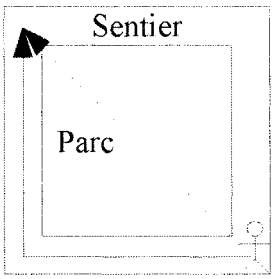


Figure 2.25: Illustration de la distance géodésique

De la même façon, sur une surface formée de triangles, les distances entre deux points doivent être calculées comme si la personne marchait sur la surface en ligne droite entre ces deux points sans se limiter à suivre les arêtes des triangles. À la figure 2.26, il ne devrait donc pas y avoir de différence entre la distance qui sépare le coin A du coin C et la distance qui sépare le coin B du coin D de la figure ci-contre, ce qui n'est évidemment pas le cas si les distances sont calculées à partir des arêtes des triangles.

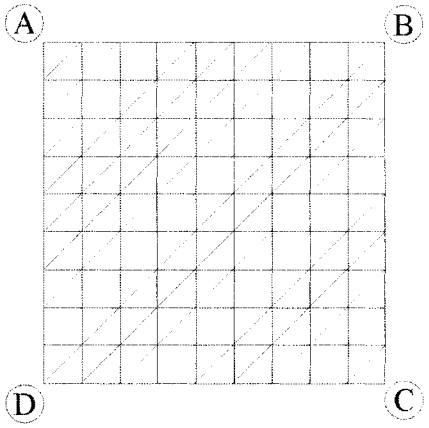


Figure 2.26: Exemple de triangulation d'un plan

Sur la figure 2.27, on présente la différence entre le calcul des distances géodésiques et celui des distances de Manhattan sur une surface plane de 50 par 50 unités divisée en triangles de la façon présentée à la figure 2.26. Les couleurs de la figure 2.27 représentent la distance proportionnelle à partir du coin inférieur gauche. Le contraste des figures a été augmenté pour permettre de mieux distinguer les différences entre les deux types de distances.

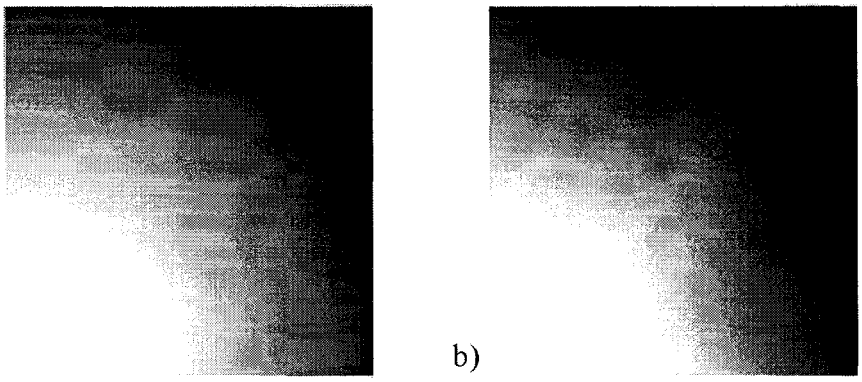


Figure 2.27: a) Distances géodésiques VS b) distances de Manhattan

On voit clairement que le calcul correct des distances géodésiques (à gauche sur la figure) fournit des résultats formant un patron circulaire, ce qui est désiré, tandis que si les distances sont calculées en suivant les arêtes des triangles (distances de Manhattan, à droite sur la figure), on n'obtient pas réellement la forme circulaire attendue. Il est à noter que le long de la diagonale, on voit apparaître une droite plus pâle puisqu'en suivant les arêtes diagonales des triangles, on peut se rendre à un point situé le long de la diagonale par le chemin le plus court, ce qui se reflète sur la figure.

Donc, bien que le calcul des distances de Manhattan par Dijkstra soit, en pratique, beaucoup plus simple (et plus rapide), il a été établi que le calcul des distances géodésiques est nécessaire pour obtenir une méthode indépendante de la distribution des triangles sur la surface. Il ne reste qu'à présenter comment calculer ces distances.

Pour calculer les distances géodésiques, on utilise le *Fast Marching Method* (Kimmel et Sethian, 1998). Voici l'algorithme général :

```
- Initialiser tous les points à l'état LOIN
- Initialiser le point courant à l'état ACTIF
- Initialiser tous les voisins du point courant à l'état PROCHE

TANT QUE la distance géodésique maximale voulue n'est pas atteinte
      - Choisir le point PROCHE le plus près du point courant
      - Mettre ce point à l'état ACTIF
      - Mettre les voisins de ce point à l'état PROCHE
      - Calculer les distances pour les points à l'état PROCHE à l'aide
        des points à l'état ACTIF seulement
```

Cette méthode permet une progression strictement croissante des distances et permet de mettre un seuil au dessus duquel il ne sera pas nécessaire de calculer les distances. Il est à noter que seuls les points à l'état ACTIF ne pourront plus changer de valeur au cours de l'algorithme, puisque les points à l'état PROCHE peuvent toujours se réajuster au besoin.

L'article de Kimmel et Sethian (1998) présente l'adaptation du *Fast Marching Method* pour des grilles orthogonales, des domaines triangulés planaires, des domaines triangulés sans triangles obtus et finalement pour des domaines triangulés quelconques. Les explications qui suivent mentionneront d'abord la méthode adaptée pour les domaines triangulés sans triangles obtus, puis l'adaptation nécessaire pour les triangles obtus.

Essentiellement, pour chaque triangle, il y a deux possibilités : soit le point PROCHE à calculer se trouve dans un triangle possédant un point ACTIF, soit le triangle en possède deux, puisqu'un point PROCHE à calculer se trouve forcément dans au moins un triangle possédant un point ACTIF. Maintenant, s'il n'y a qu'un point ACTIF, il n'est pas possible de faire mieux que d'ajouter la distance calculée du point ACTIF à la distance qui sépare les deux points pour calculer la distance du point PROCHE.

Cependant, s'il y a deux points ACTIF dans le triangle, il est alors possible de procéder aux calculs. Prenons comme référence la figure suivante dans laquelle les cercles pleins identifient les points ACTIF et les cercles vides identifient les points PROCHE. Tous les points voisins d'un point ACTIF sont PROCHE, donc il n'y a que deux possibilités de triangles possédant à la fois un point ACTIF et un point PROCHE.

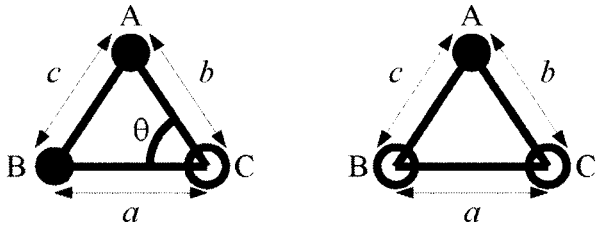


Figure 2.28: Triangles avec deux (à gauche) ou un (à droite) point(s) ACTIF

En utilisant la notation D(x) comme étant la distance entre le point x et le point courant à partir duquel les distances sont calculées et en prenant comme exemple la figure 2.28, il est possible d'établir que pour le triangle de droite, les calculs donnent :

$$D(B) = D(A) + \mathcal{F}c \qquad \text{et} \qquad D(C) = D(A) + \mathcal{F}b$$

où $\mathcal{F}$ est la fonction de vitesse et peut être remplacée par un (1) si aucun poids n'est associé aux différents triangles, ce qui est le cas pour l'utilisation que l'on fait de cette méthode dans le cadre de la méthode PF.

Maintenant, si le triangle courant contient les mêmes états que le triangle de gauche, on doit commencer par trouver t à l'aide de l'équation quadratique suivante :

Soit $u = D(B) - D(A)$:

$$(a^2 + b^2 - 2ab\cos\theta)t^2 + 2bu(a\cos\theta - b)t + b^2(u^2 - \mathcal{F}^2a^2\sin^2\theta) = 0 \qquad (2.30)$$

où t est la valeur recherchée. Ceci permet d'obtenir facilement $\pm t$. La solution t doit satisfaire aux conditions :

$$u < t \qquad \text{et} \qquad a\cos\theta < b(t-u)/t < a/\cos\theta$$

Ceci permet de mettre à jour la distance géodésique du point C selon :

```
SI u < t ET a cosθ < b(t-u)/t < a/cosθ ALORS
     D(C) = min{ D(C) , t+D(A) }
SINON
     D(C) = min{ D(C), bℱ+D(A) , aℱ+D(B)}
```

Ces dernières conditions permettent de vérifier que la solution fait bien partie du triangle, donc que la solution est mathématiquement valide. S'il n'existe pas de solution t, alors le calcul se fait par une somme, de la même manière que pour le triangle précédent. Puisque le triangle possède deux points ACTIF, il faut cependant faire le minimum des deux possibilités. Aussi, il est à noter que le minimum de D(C) inclut toujours D(C) lui-même, puisqu'il est toujours possible qu'une solution trouvée sur un triangle différent soit meilleure. Pour de plus amples détails sur l'origine de ces équations, l'article de Kimmel et Sethian (1998) fournit le développement mathématique des équations qui ne sera pas inclus ici pour éviter d'alourdir les explications.

Maintenant, pour ce qui est des triangles obtus, une adaptation simple peut être faite. Le but est que la triangulation fasse en sorte que l'angle θ de la figure 2.28 ne soit pas obtus. Ceci permet de faire de la mise à jour du point C à partir de ce triangle. Pour les triangles obtus, il faut créer des triangles virtuels non-obtus afin d'effectuer les mises à jour. Afin de créer ces triangles virtuels, il faut trouver un point qui permettra de mettre à jour le point C. Pour ce faire, il faut s'assurer que ce point fasse en sorte que l'angle θ du triangle virtuel ne soit pas obtus. Le but est donc de trouver un point répondant à ces conditions. Par exemple, à la figure 2.29, les deux lignes pointillées partant du point C montrent la zone dans laquelle le point à trouver doit se situer. Cette zone est déterminée à partir de deux angles droits partant des deux segments formant l'angle obtus.

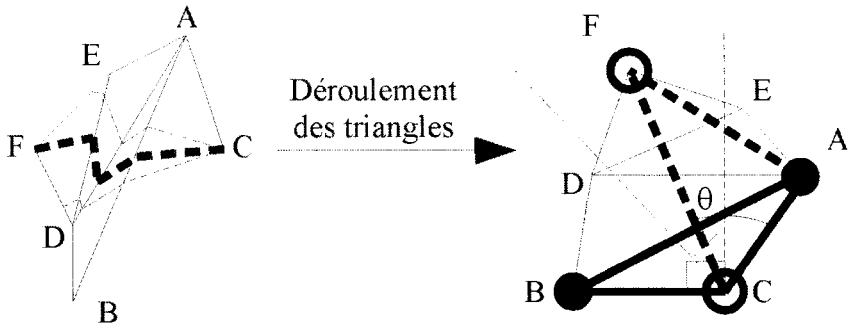


Figure 2.29: Création d'un triangle virtuel

À tour de rôle, il suffit de « dérouler » les triangles pour qu'ils se retrouvent dans le même plan que le triangle ABC. Donc, sur la figure 2.29, le triangle ABD effectue donc une rotation autour de l'axe AB. Le point D n'étant pas valide, on détermine que la zone valide passe entre les points A et D et on déroule le triangle ADE. On procède ainsi jusqu'à ce qu'un point soit trouvé ou qu'aucun triangle ne soit disponible. Finalement, sur la figure 2.29, on trouve le point F qui permettra de former les triangles virtuels valides ACF et BCF. Pour mettre à jour le point C, il sera donc possible de se servir du point F (mais pas l'inverse). Si le point F est ACTIF au moment de la mise à jour du point C, la méthode décrite précédemment pour les triangles non-obtus pourra être utilisée.

2.4.1.3.c Calcul et choix des point fingerprints

Une fois que les distances géodésiques sont calculées, il devient possible de construire les *point fingerprints*. Pour ce faire, il ne s'agit que de choisir un certain nombre de distances (ou rayons) auxquelles on désire dessiner des cercles autour d'un point. Pour une surface parfaitement plane, tous les points d'une surface se trouvant à la même distance géodésique d'un point forment un cercle. Au fur et à mesure que la surface se déforme, le cercle se déforme aussi, épousant la forme de la surface. On projette alors ces cercles sur le plan possédant la même normale que le point courant et ceci fournit un cercle plus ou moins déformé par le relief de la surface.

À la figure 2.30, on pourrait s'imaginer que l'ensemble de courbes de gauche a été déformé par une bosse ou un trou dans le relief de la surface. Aussi, si, par exemple, 5 rayons allant de 1 à 5 unités de longueur sont choisis, chaque point pourra être représenté par 5 courbes qui représentent la déformation locale pour le point courant.

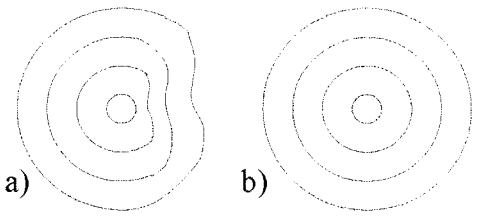


Figure 2.30: Exemple de *point fingerprints* a) avec relief et b) sans relief

Pour sélectionner les *point fingerprints* les plus significatifs, il suffit de choisir les courbes qui varient le plus. En effet, sur un plan, un point obtiendrait un *point fingerprint* formé de courbes parfaitement circulaires, donc de rayon égal en tout point. Si le relief est plus grand, alors, en certains endroits, le rayon de la courbe entourant le point variera. Il suffit alors de choisir les courbes possédant le plus grand rapport entre les rayons maximum et minimum.

2.4.1.3.d Corrélation des point fingerprints

Une fois que chaque point de chaque surface possède une représentation sous forme de *point fingerprint*, il est possible de procéder à l'appariement des points en effectuant une corrélation entre ces *point fingerprints*. Pour ce faire, il faut d'abord rééchantillonner chaque courbe pour obtenir un échantillonnage régulier et constant dans toutes les données. Théoriquement, les *point fingerprints* de deux points représentant le même point réel sur les deux surfaces devraient être les mêmes à une rotation près. Dans Sun 2003, on propose d'effectuer la corrélation entre les normales des points obtenus du rééchantillonage. La formule utilisée est la suivante :

$$R_{ij} = \min_{l=1}^{K} \sum_{m=1}^{M} \sum_{k=1}^{K} \left(n_{1,i,m,k} \cdot n'_{1,i} - n_{2,j,m,k+l} \cdot n'_{2,j} \right)^2 \qquad (2.31)$$

où $n_{1,i,m,k}$ est la normale du $k^{\text{ième}}$ point de la $m^{\text{ième}}$ courbe du $i^{\text{ième}}$ *point fingerprint* pour la première surface et $n'_{1,i}$ est la normale du point central du $i^{\text{ième}}$ *point fingerprint* pour la première surface. $n_{2,j,m,k}$ et $n'_{2,j}$ sont semblables, mais pour la seconde surface. Le $i^{\text{ième}}$ point de la première surface correspond donc au $j^{\text{ième}}$ point de la seconde surface si :

$$j = \arg\min_k R_{ik}$$

et que R_{ij} est sous un seuil prédéterminé. On mentionne aussi que la variation dans la forme des courbes peut aussi être utilisé pour confirmer l'appariement des points.

2.4.2 Conclusion de la revue de littérature

Jusqu'à maintenant, les revues de la littérature des méthodes de segmentation et de recalage d'images ont été présentées. À partir des différentes méthodes et techniques expliquées, il devrait être possible de résoudre les problématiques exposées plus tôt et d'automatiser les traitements manuels non reproductibles (détourage de la région d'intérêt et utilisation de la toile pour le recalage). Dans le contexte des méthodes non effractives de suivi et de diagnostic de la scoliose présentées plus tôt, ces méthodes devront être adaptées et optimisées de façon à fournir des solutions efficaces aux différentes problématiques. La sous-section suivante présente en détail les objectifs de ce mémoire.

2.5 Objectifs

2.5.1 Objectif général

Le but ultime de ce projet est de développer un outil de suivi non-effractif des déformations musculo-squelettiques. Pour ce faire, un outil de reconstruction 3D automatique reproductible et utilisable en clinique du tronc humain à partir d'images acquises par des numériseurs 3D de la compagnie InSpeck doit être développé. Pour réaliser cet objectif général, deux objectifs spécifiques doivent être atteints en l'occurence :

- Trouver une région d'intérêt (tronc) sur une image bidimensionnelle.
- Recaler deux ou plusieurs surfaces (différentes vues du tronc) partiellement correspondantes.

2.5.2 Objectifs spécifiques

Pour la segmentation et la détection des régions d'intérêt, il faudra :

- Trouver un moyen de tirer partie de toutes les informations à notre disposition.
- Quantifier expérimentalement certains paramètres pour que l'algorithme ait du sens dans le présent contexte.
- Trouver une façon logique de fournir une approximation initiale.
- Établir des critères permettant de choisir quels segments font partie de la région d'intérêt, ainsi que d'autres critères permettant d'éliminer les bras et la tête d'une personne de la région d'intérêt si le tronc est la seule partie de l'image qui nous intéresse.

Pour le recalage des différentes parties du tronc, il faudra :

- Déterminer quels algorithmes peuvent être appliqués même si les surfaces ne sont que partiellement correspondantes.
- Vérifier si les différents algorithmes peuvent être entièrement automatisés ou non, selon le cas.

Chapitre 3 : Méthodologie

Afin d'automatiser complètement le processus de reconstruction 3D du tronc, il faut donc compléter un ensemble de procédures. La méthodologie adoptée est illustrée par le diagramme de la figure 3.1 avec les parties importantes en italique.

Ce chapitre se subdivise en plusieurs sections.

La première section traitera du développement des méthodes de segmentation d'image. Cette section présente diverses adaptations des algorithmes pour optimiser leurs performances dans le présent contexte. Ces améliorations incluent, entre autres, la définition d'un nouveau système de représentation d'une image et l'élaboration d'un algorithme permettant le choix automatique du ou des segments voulus d'une image, donc du choix automatique de la région d'intérêt.

La seconde section portera sur le recalage des surfaces. Cette section présente divers problèmes rencontrés avec la définition initiale de la méthode PF ainsi que les moyens qui ont été pris pour résoudre ces problèmes et aussi pour améliorer la qualité du recalage obtenu. Ces solutions incluent, par exemple, une nouvelle méthode de corrélation croisée et plusieurs nouvelles méthodes permettant d'améliorer la robustesse de l'appariement des points.

Finalement, la dernière section établira la méthode de validation des résultats des algorithmes qui seront présentés dans ce chapitre.

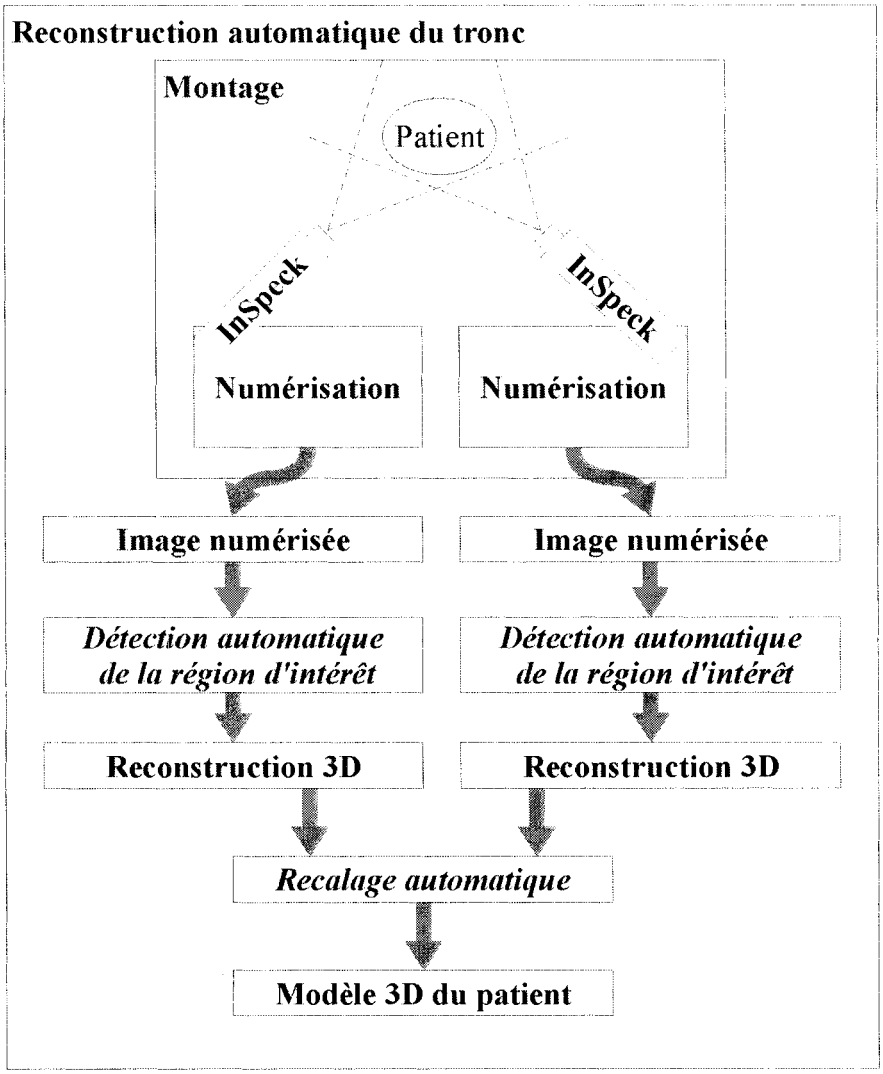


Figure 3.1: Reconstruction automatique du tronc

3.1 Détection de la région d'intérêt

Rappelons que, pour séparer l'image en régions distinctes, sans utiliser de connaissances *a priori*, les critères de segmentation qu'on peut utiliser sont les suivants :

- Chaque région tend à avoir une certaine homogénéité d'apparence.
- L'« apparence » est différente d'une région à une autre. Donc il y a des discontinuités d'apparence aux frontières entre les régions.

Dans la littérature, la majorité des algorithmes de segmentation basés sur des modèles statistiques sont conçus précisément pour répondre à ces types de critères (homogénéité intra-classe, discontinuité inter-classe). Pour cette raison, les approches statistiques ont été explorées pour segmenter les images de topographie de surface.

3.1.1 Objectif et contexte

Le but de cette section est de réaliser un algorithme de segmentation autonome sans *a priori* pour déterminer les régions d'intérêt sur des images de topographie de surface utilisées pour le diagnostic clinique de la scoliose. Les images à segmenter contiennent un(e) patient(e) en avant plan ainsi qu'un certain nombre d'objets en arrière-plan. Le contenu de l'arrière plan de l'image varie beaucoup selon l'environnement.

L'objectif primaire de cette partie du projet est de séparer l'image, sans intervention humaine, en régions simples qui représentent les différents objets qui pourront, par la suite, être classés comme avant-plan ou arrière-plan. Comme mentionné plus tôt, le problème de classification des régions (avant/arrière-plan) nécessite une segmentation préalable de l'image. Le processus de segmentation consiste à séparer l'image en des régions les plus grandes possible, sans qu'il y ait de régions qui chevauchent avant-plan et arrière-plan.

Aussi, obtenir une segmentation d'image sans intervention humaine n'est pas chose facile. Dans la réalité, plusieurs régions de l'image peuvent avoir des contours flous ou des couleurs semblables. Dans la segmentation sans connaissances *a priori*, il est rare que le nombre de régions distinctes soit connu. Le but est donc aussi de segmenter l'image en un nombre optimal de régions distinctes qui permettront de bien déterminer la région d'intérêt.

3.1.2 Technique de segmentation

Cette section présente en détail les ajustements effectués à l'algorithme du PLIC.

3.1.2.1 Extraction des données et prétraitement des images de topographie de surface

Les images de topographie de surface acquises par les caméras de InSpeck inc. sont multispectrales. Une image TS non reconstruite est composée de quatre composantes :

- Trois composantes de texture (R, G et B : photographie numérique RGB de la scène);
- Une composante de phase générée par une technique d'interférométrie. Cette composante représente le relief de la scène (Le principe des images de phase a déjà été discuté au chapitre 1 à la section 1.2.1).

Avec la composante de phase on peut profiter d'une propriété intrinsèque aux instruments d'acquisition en topographie de surface : les objets en arrière-plan (hors du volume d'acquisition efficace de la caméra) ont tendance à avoir une phase plus bruitée. Afin de profiter de cette propriété, une étape de prétraitement a été introduite et permet de distinguer les zones plus bruitées des zones moins bruitées. Cette étape consiste à calculer, pour chaque pixel ayant une valeur de phase P_i (où $P_i \in [0, 1[$), une valeur d'homogénéité de phase HP_i :

$$HP_i = \sum_{j \sim i} (P_i - P_j)^2 \tag{3.1}$$

où $j \sim i$ signifie qu'on effectue la sommation sur les 8 pixels j constituant le voisinage du pixel i. Le résultat est donc la somme de la différence au carré de la valeur de phase d'un pixel avec celle de chacun de ses voisins. Il est à noter que, pour tenir compte du déroulement de phase, la condition suivante a été implémentée :

si $P_i - P_j > 1/2$, alors on le remplace par $1 - (P_i - P_j)$.

Aussi, les bordures et les coins n'ayant respectivement que 5 ou 3 voisins ont été ramenés à la même échelle que le reste de l'image avec une multiplication par 8/5 ou 8/3 afin que toutes les données soient du même ordre de grandeur. Les pixels situées aux coins et aux bordures des images n'étant pas d'une très grande importance dans la grande majorité des cas, l'interpolation linéaire est amplement suffisante.

L'homogénéité de phase est plus utile que la phase brute parce qu'il s'agit d'une caractéristique discriminante entre les objets d'avant et d'arrière-plan. En intégrant la composante d'homogénéité de phase à l'image à segmenter, on apporte une information qui aidera à éviter de créer des régions qui chevauchent l'avant et l'arrière-plan. De plus, le résultat du calcul d'homogénéité fut filtré par un filtre médian pour adoucir la composante d'homogénéité de phase. La figure 3.2 présente un exemple de cette composante. On y voit clairement que l'homogénéité de phase est une représentation du niveau de bruit de chaque région de l'image. On y voit aussi, par l'ombre de la patiente en arrière-plan, que la source de lumière et la caméra ne sont pas coaxiales, ce qui est une propriété de la triangulation mentionnée à la sous-section 2.2.3.1.b.

Image de phase originale Homogénéité de phase

Figure 3.2: Exemple d'homogénéité de phase

En ce qui concerne les composantes de texture, il faut tenir compte du fait qu'un objet de couleur homogène ne produit pas nécessairement une région de couleur homogène sur l'image à cause des conditions d'éclairages qui peuvent varier entre différentes parties de l'objet. Selon le modèle de réflexion diffuse Lambertien, la lumière émise par une surface est modulée par la lumière incidente proportionnellement au cosinus de l'angle formé par le rayon d'incidence et la normale à la surface (Trucco et Verri 1998).

Image RGB originale Image UV (U en vert, V en bleu)

Figure 3.3: Exemple d'isolation du chroma par conversion YUV

En assumant que toutes les sources de lumière dans l'environnement d'acquisition ont le même spectre d'émission et que les surfaces observées sont Lambertiennes (réflexion diffuse pure) et homogènes, on arrive à la conclusion que seulement l'intensité de la lumière émise par la surface dépend du flux de lumière incident. Autrement dit, le chroma (spectre de couleur normalisé) est invariant à l'intensité d'éclairage. Pour tenir compte de cette réalité, les images ont été converties dans l'espace de couleur YUV et seules les composantes U et V qui représentent le chroma ont été conservées (exemple à la figure 3.3). Avec cette nouvelle représentation

des composantes de texture, on obtient des régions plus ou moins homogènes pour les objets de couleur homogène.

3.1.2.1.a Nouveau système de représentation d'une image

Pour résumer ce qui vient d'être présenté, à partir des quatre composantes de départ [R, G, B, Phase], on obtient des images formées des trois composantes [Homogénéité de phase filtrée, U, V] qui pourront être segmentées. La figure 3.4 présente la progression de ce changement. De l'image RGB, on obtient les composantes U et V (figure 3.4c, U en vert, V en bleu). De l'image de phase, on obtient la composante d'homogénéité de phase (figure 3.4d, en rouge). Ces trois composantes se combinent pour former l'image finale (figure 3.4e, U en vert, V en bleu, Homogénéité de phase filtrée en rouge).

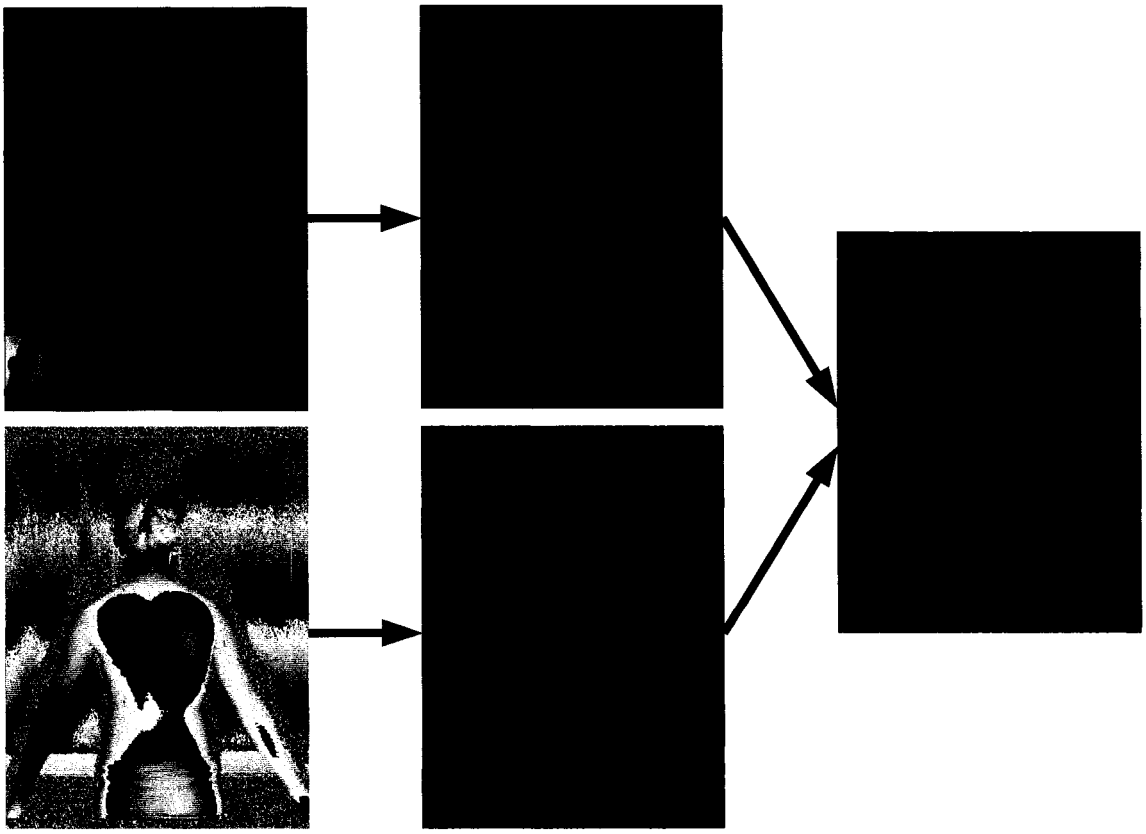


Figure 3.4: Nouveau système de représentation d'une image

3.1.2.2 Vue d'ensemble de la méthode de segmentation

Voici une vue d'ensemble de la méthode de segmentation qui sera présentée dans les pages suivantes.

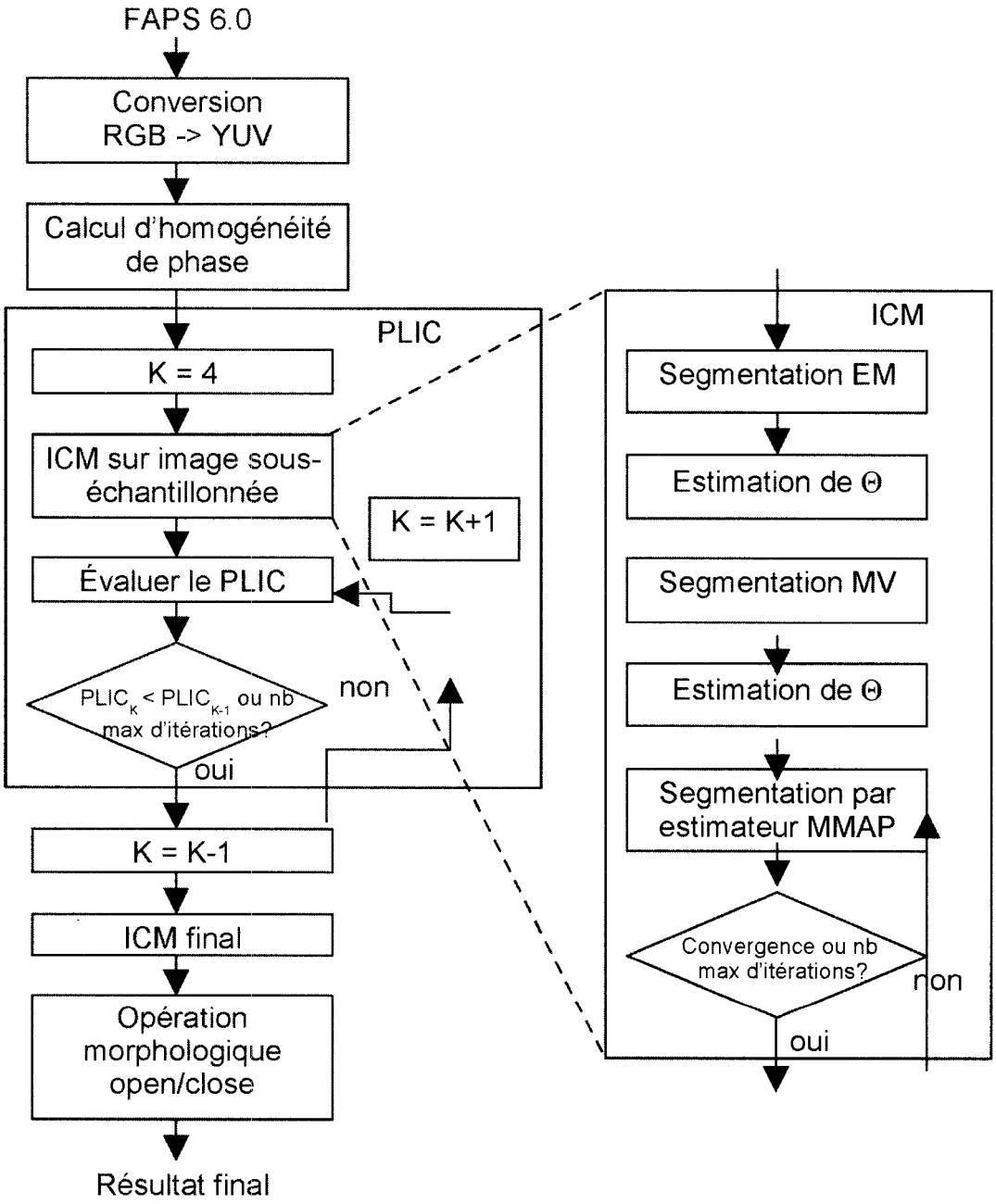


Figure 3.5: Diagramme fonctionnel de la technique de segmentation

3.1.2.3 La fonction de vraisemblance

Pour ce projet, l'approche paramétrique avec un modèle de distribution gaussienne fut choisie. Avec cette approche, chaque classe est caractérisée par un vecteur de paramètres θ_m. Dans le cas où on utilise la distribution gaussienne multivariée, le contenu de θ_m est un vecteur moyenne μ_m et une matrice de covariance Σ_m. La densité de probabilité de Y_i peut donc être exprimée de la façon suivante :

$$f(Y_i|X_i = m, \Theta) = \begin{Bmatrix} f(Y_i|\theta = \theta_1) & si & X_i = 1 \\ \vdots & & \vdots \\ f(Y_i|\theta = \theta_K) & si & X_i = K \end{Bmatrix}$$

$$\Theta = \{\theta_1, \ldots, \theta_K\}$$

$$\theta_m = \{\mu_m, \Sigma_m\}$$

$$f(Y_i|\theta) = \frac{1}{(2\pi)^{\Omega(\mu)/2}\sqrt{|\Sigma|}} e^{\frac{1}{2}(Y_i-\mu)^T \Sigma^{-1}(Y_i-\mu)}$$

Note : Ω représente l'opérateur de cardinalité.

Étant donné que le nombre d'états possibles pour X_i est fini et de valeur K, il est facile de résoudre l'estimateur $\hat{X}_{MV,i}$: il suffit d'évaluer $f(Y_i|X_i = m, \Theta)$ pour toutes les valeurs de m de 1 à K et de retenir la valeur de m qui donne la meilleur vraisemblance.

3.1.2.4 L'approximation initiale

Bien que la technique de segmentation MV (voir figure 3.9) puisse fournir une approximation initiale de la segmentation, elle ne peut pas servir à estimer Θ. De plus, la segmentation MV requiert une connaissance *a priori* de Θ. Dans l'article original sur l'ICM, Besag (1986), on ne suggère aucune technique d'approximation initiale des paramètres. L'auteur suggère que l'utilisateur fournisse des approximations grossières des paramètres des classes et que l'algorithme ICM saura converger vers la bonne solution. Cela cadre très mal dans un contexte où il n'y a aucune connaissance *a priori*.

Afin de fournir à l'algorithme une approximation initiale raisonnable, il est important que l'algorithme choisi soit lui-même rapide et qu'il favorise la convergence de l'algorithme ICM le plus rapidement possible. Le choix s'est arrêté sur un algorithme de segmentation EM (*Expectation-Maximisation*) à cause de sa simplicité et de son caractère déterministe. La simplicité favorise la rapidité d'exécution et le caractère déterministe permet une bonne reproductibilité de l'algorithme, peu importe l'environnement.

L'implantation de la segmentation EM a été réalisée telle que décrite dans Carson 1999 avec quelques ajustements pour améliorer les résultats avec les images de patients. Tout d'abord, pour la toute première approximation de Θ, on suggère de segmenter l'espace géométriquement selon un modèle constant pour toutes les images. La proposition de Carson 1999 est décrite à la figure 3.6.

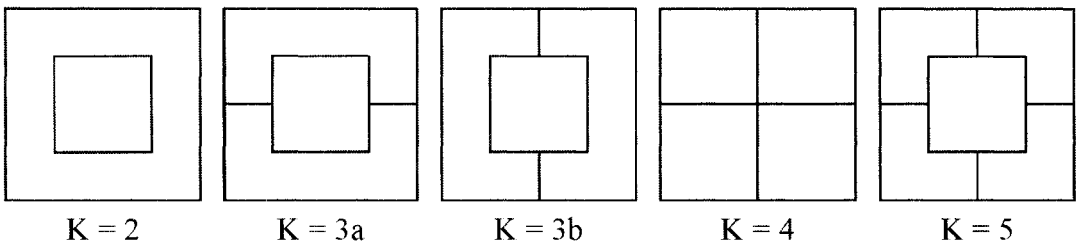


Figure 3.6: Fenêtres pour l'initialisation de Θ de la segmentation EM

Or, nous supposons que le patient est habituellement approximativement au centre de la photo. Aussi, le nombre idéal de segments dépasse presque toujours 5 (souvent entre 7 et 9 à cause des objets en arrière-plan). Les segments « cheveux », « vêtements », « arrière-plan » et « peau » font en sorte que l'image se divise en général au moins en 4 segments, ce pourquoi notre K initial sera posé à 4 plutôt qu'à 2 comme le suggèrent Stanford et Raftery (2002). Lors des examens de suivi, les patients atteints de la scoliose sont souvent dans l'une des deux positions illustrées à la figure 3.7.

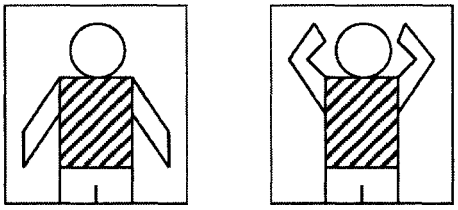


Figure 3.7: Positions approximatives des patients dans les images.

Dans les deux cas, la région d'intérêt de ces images peut donc être considérée comme étant approximativement rectangulaire et centrée. La nouvelle proposition de fenêtres est donc décrite à la figure 3.7.

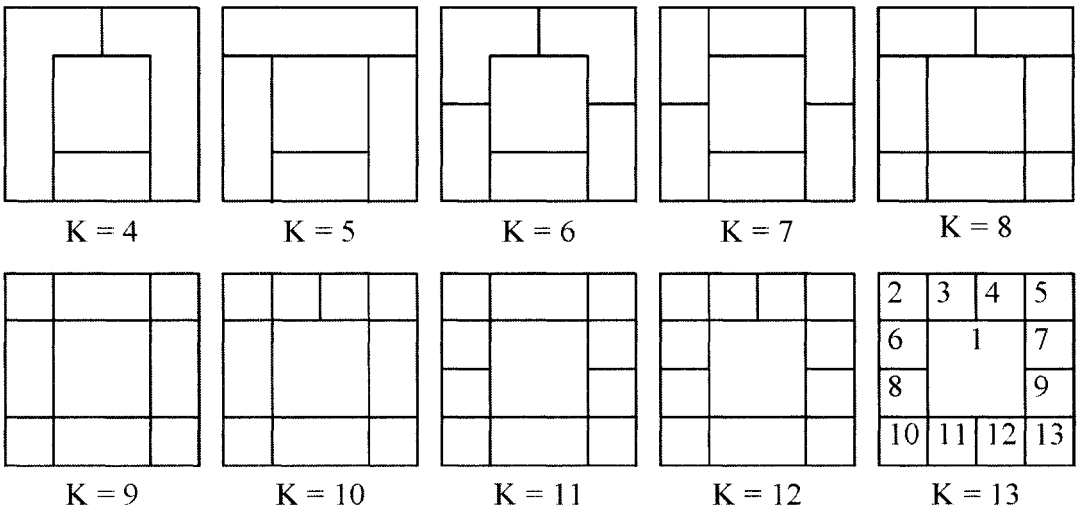
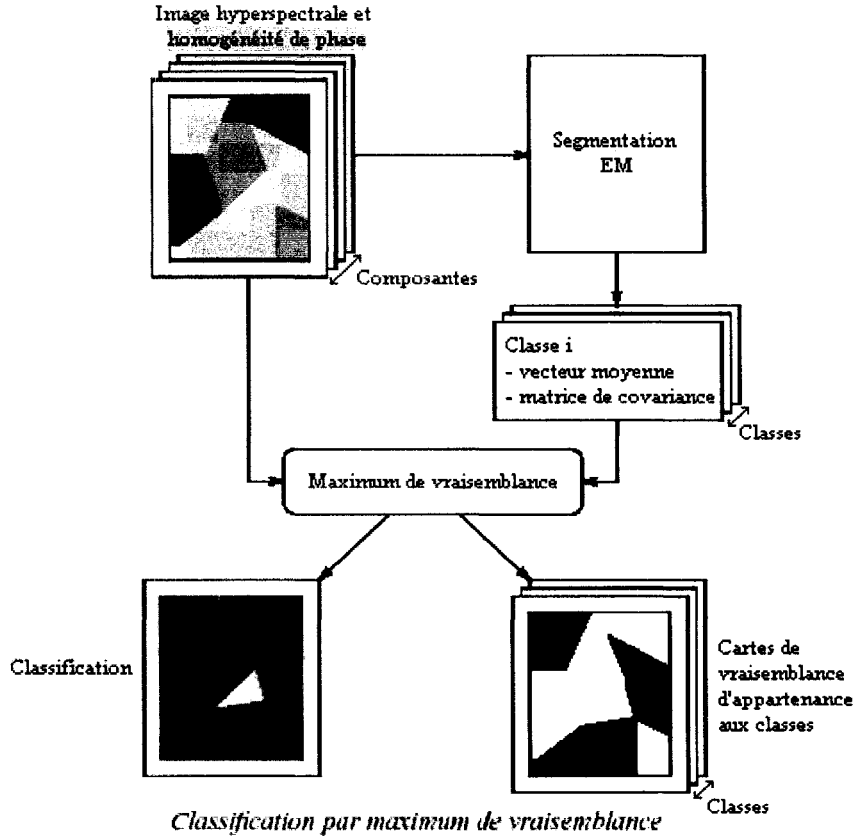


Figure 3.8: Fenêtres modifiées pour l'initialisation de Θ de la segmentation EM

Sur la figure 3.8, pour K = 13, sont numérotées les 13 régions avec lesquelles on peut former n'importe laquelle des approximations de la cette figure, moyennant une fusion de certaines régions. Normalement, la région 1 devrait contenir le torse de la personne. Les régions 11 et 12 contiennent habituellement la culotte ou le pantalon, ce qui explique que ces régions soient fusionnées quand K est plus petit que 13. Les régions 6,7,8 et 9 contiennent les bras et une partie de l'arrière-plan. Les régions 3 et 4 contiennent la tête et une partie de l'arrière-plan. Les régions 2,5,10 et 13 devraient contenir surtout l'arrière-plan. L'arrière-plan se compose parfois d'objets divers et même d'autres personnes, dans certains cas, ce qui complique parfois la segmentation.

Il est à noter que les approximations initiales ont été déterminées en tentant de conserver les zones semblables fusionnées et en respectant la symétrie, puisque l'image est approximativement symétrique.



Classification par maximum de vraisemblance

Figure 3.9: Principe de segmentation au sens du maximum de vraisemblance

3.1.2.5 Le paramètre ϕ du modèle de Potts

Comme mentionné plus tôt, le paramètre ϕ indique la « force » de la tendance d'homogénéité imposée par le modèle de Potts. Plus précisément :

Par rapport à l'état m de X_i, ses voisins ont tendance à être : $\begin{cases} \phi > 0 : \textit{Semblables} \\ \phi < 0 : \textit{Différents} \\ \phi = 0 : \textit{Indépendants} \end{cases}$

Dans le cas présent, il est préférable que les pixels aient tendance à être semblables, car les régions à découper sont plutôt lisses, unies et larges. Il est donc clair qu'une valeur de $\phi > 0$ s'impose.

Selon Besag 1986, ICM peut estimer la valeur de ϕ à chaque itération, mais aucune forme explicite de cette méthode n'est fournie avec la description de ICM. Besag mentionne seulement qu'il serait possible de le faire en maximisant la pseudo-vraisemblance de ϕ selon l'estimation actuelle de X :

$$\hat{\phi} = \arg\max_{\phi} \prod_{i \in \{1 \ldots N\}} P\left(X_i \middle| \hat{X}, \phi\right) \tag{3.2}$$

En posant le modèle de Potts tel que défini plus tôt, on obtient :

$$\hat{\phi} = \arg\max_{\phi} \prod_{i \in \{1 \ldots N\}} \frac{\exp\left(\phi V\left(X_i, m\right)\right)}{\sum_{t=1}^{K} \exp\left(\phi V\left(X_i, t\right)\right)} \tag{3.3}$$

Pour simplifier l'expression, prenons le log-vraisemblance :

$$\hat{\phi} = \arg\max_{\phi} \sum_{i=1}^{N} \left(\log\left(e^{\phi V(X_i,m)}\right) - \log\left(\sum_{\ell=1}^{K} e^{\phi V(X_i,\ell)} \right) \right)$$
$$= \arg\max_{\phi} \sum_{i=1}^{N} \left(\phi V(X_i,m) - \phi - \log\left(\sum_{\ell=1}^{K} e^{V(X_i,\ell)} \right) \right)$$
$$= \arg\max_{\phi} \phi \sum_{i=1}^{N} \left(V(X_i,m) - 1 \right)$$

Résultat : Cet estimateur est divergeant avec cette version du modèle de Potts. Par conséquent, il faudra déployer une autre stratégie pour fixer une valeur de ϕ expérimentalement.

3.1.2.6 Optimisation de la performance

Pour accélérer les calculs du PLIC, un sous-échantillonnage de l'image a été effectué afin d'utiliser moins de points. Aussi, le nombre d'itérations a été limité à 12 pour le PLIC (Lorsque l'on dépasse largement 10 segments, la segmentation peut devenir farfelue) et à 10 pour ICM au cas où la segmentation convergerait trop lentement.

Comme critère d'arrêt de l'algorithme ICM, la spécification choisie est que l'image obtenue doit être au moins à 99% identique à l'une des deux itérations précédentes. La comparaison est effectuée entre les deux itérations précédentes au cas où l'algorithme oscillerait entre deux réponses.

3.1.3 Choix des paramètres

Afin de déterminer les valeurs appropriées du modèle dans le contexte de notre application, il a été nécessaire de procéder à des tests utilisant quelques types d'images différents. Plus spécifiquement, il fallait :

- Déterminer une valeur acceptable pour ϕ du modèle de Potts;
- Vérifier que le PLIC est maximal pour un nombre de classes K optimal;
- Tester les algorithmes sur des images de niveaux de difficultés croissants.

Il faut comprendre que les comparaisons sont purement qualitatives et que la cible optimale pour le PLIC est un intervalle plutôt qu'une valeur exacte. Avec cette philosophie en tête, une valeur optimale de ϕ a été assignée pour une image de test. Ensuite, la valeur fut raffinée par des essais sur une banque d'images dans le but de trouver une solution globalement optimale. Afin d'éviter que les résultats soient biaisés par la nature spécifique du contenu des images de test, des images très différentes ont été utilisées.

Dans le chapitre « Résultats et discussion », ces images et les résultats de leur segmentation seront présentés.

3.1.3.1 Implantation des algorithmes de segmentation

Les algorithmes de segmentation décrits précédemment ont été implantés afin d'en évaluer les performances sur des images réelles et/ou synthétiques. L'implantation des différentes méthodes ont été faites entièrement en MATLAB.

3.1.4 Choix des segments de la région d'intérêt

Une fois l'image segmentée, il reste encore le choix de la région d'intérêt à faire. Ce choix réside dans l'ajout de certains critères qui seront choisis selon les connaissances *a priori* anatomiques de la région d'intérêt. Par exemple, la partie centrale contient normalement la région d'intérêt. En sachant cela, le segment occupant la majeure partie du centre de l'image sert de point de départ comme région d'intérêt.

Par la suite, si la région d'intérêt est composée de plusieurs segments distincts, on joint à cette région les segments permettant de lier les segments de la région d'intérêt. L'intérêt de cette étape est d'ajouter certains segments distincts, comme par exemple un soutien-gorge, à la région d'intérêt afin que cette région possède un seul segment au lieu de plusieurs petits segments. Une fois la région déterminée, il reste le retrait des bras et du cou à effectuer. Pour retirer le cou, on recherche simplement une région significativement plus mince que le torse située en haut du torse. Pour couper les bras, on balaye l'image horizontalement de haut en bas pour détecter les zones de concavités. Lorsqu'une ligne horizontale contient deux segments distincts de la région d'intérêt, on trouve la position du premier bras et on trouve ensuite le second bras lorsque trois segments distincts de la région d'intérêt se retrouvent sur la même ligne. Si les deux bras sont exactement à la même hauteur, le nombre de segments passera directement de 1 à 3 (voir figure 3.10). Enfin, le résultat est une image binaire indiquant si, oui ou non, chaque pixel de l'image fait partie de la région d'intérêt. Il suffit de faire simplement le contour de l'image binaire pour obtenir le résultat désiré, soit le contour de la région d'intérêt.

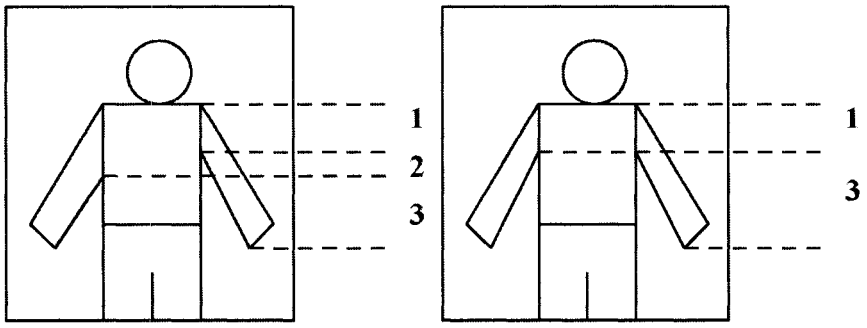


Figure 3.10: Nombre de segments horizontaux distincts

3.2 Recalage des surfaces

Après avoir complété le détourage automatique des régions d'intérêt sur les images, la reconstruction automatique des surfaces peut être effectuée. Le système fournit un nombre de surfaces partielles correspondant au nombre de caméras utilisées. Ces surfaces doivent être recalées afin de reconstruire une surface globalement cohérente. En minimisant le nombre de capteurs, la redondance entre les informations contenues dans les images se trouve aussi minimisée, ce qui rend d'autant plus difficile le recalage étant donné l'absence d'informations précises sur les positions des appareils. Le choix de la zone d'intérêt sur laquelle est basée la procédure de recalage devient donc un choix critique directement relié à la qualité de la surface obtenue. Il est à noter que les surfaces recalées au cours de ce mémoire contiennent les surfaces avant du tronc, même si la maladie à l'étude est la scoliose. Ceci s'explique par le fait que la déformation de la colonne vertébrale entraîne aussi souvent une déformation des côtes et que cela est visible sur la partie avant du tronc. Le recalage de toutes les parties du tronc a donc une grande importance clinique afin d'effectuer un suivi correct de la maladie.

3.2.1 Algorithme général de la méthode de recalage

```
POUR chaque surface
      POUR chaque point
            - Calculer les distances géodésiques entre le point courant
              et ses voisins situés à moins d'une certaine distance
            - Interpoler les points à plusieurs distances géodésiques
              prédéterminées plus petites que la distance maximale
            - Projeter ces points interpolés sur un plan dont la
              normale est la normale du point courant
            - Recentrer les courbes sur la moyenne des points
              projetés qui forment chaque courbe
      - Choisir les courbes (point fingerprints) présentant le meilleur
        rapport entre le rayon minimal et le rayon maximal tout en
        respectant une zone d'exclusion (rayon = distance centre-point)
      POUR chaque point fingerprint sélectionné
            - Rééchantillonner chaque courbe (i.e. Chaque ensemble de
              point possédant la même distance géodésique) à angle
              constant (ce qui forme le point fingerprint final)
- Apparier les courbes grâce à une corrélation croisée circulaire
- Vérifier l'appariement en vérifiant la différence des distances entre
  chaque paire de points appariés et éliminer les appariements erronés
- Utiliser la méthode du quaternion dual pour recaler les surfaces
- Procéder à l'élimination des correspondances avant IPM, s'il y a lieu
- Raffiner le recalage avec IPM
```

3.2.2 Méthode des point fingerprints

Puisque la méthode IPM nécessite une bonne approximation de départ, une autre méthode doit être utilisée pour nous fournir cette approximation. La mise en correspondance des points n'étant pas automatisée, il faut donc trouver un moyen de faire l'appariement des points automatiquement, c'est-à-dire qu'il faut déterminer quels points de l'image A correspondent à quels points de l'image B dans la réalité. L'idée de faire l'appariement des points d'intérêt directement sur les textures a été rapidement abandonnée après quelques essais puisque :

- Les textures obtenues des caméras sont peu précises;
- La peau contient peu d'attributs distinctifs;
- Les attributs sont déformés d'une image à l'autre à cause de l'angle de caméra;

Exemple :

Figure 3.11: Déformation des attributs selon l'angle de vision

Donc, puisque les attributs sont très difficiles à détecter sur les textures, l'utilisation d'une méthode comme la méthode PF est essentielle.

3.2.2.1 Zone d'exclusion

Un problème relié à l'utilisation de la méthode PF est que le choix des points fait uniquement à partir du rapport rayon maximum/rayon minimum entraîne que si, par exemple, une zone d'une surface de 20000 points est particulièrement déformée et que l'on spécifie à l'algorithme de retourner les 100 points possédant le meilleur rapport rayon maximum/rayon minimum, les points risquent de se retrouver tous collés les uns sur les autres. Dans ce cas, il est préférable de relâcher la contrainte et d'aller chercher des points présentant des déformations un peu moindre, mais qui sont plus éloignés les uns des autres. De toute façon, les points présentant moins de déformation n'auront pas de chance de bien corréler avec les points présentant de grosses déformations.

Pour cette raison, une zone d'exclusion a été ajoutée dans le choix des points. Cette zone consiste simplement en une distance géodésique minimale à respecter entre deux points choisis. Cette distance est évidemment limitée par la distance maximale atteinte lors du calcul des distances géodésiques.

Cette zone permet donc d'éviter qu'une zone particulièrement déformée ou particulièrement bruitée prenne le monopole des points d'intérêt de la surface.

3.2.2.2 Corrélation croisée circulaire

Dans Sun 2003, on mentionne rapidement qu'il serait possible d'utiliser la forme des courbes pour effectuer l'appariement des points. Habituellement, pour effectuer une corrélation croisée (*cross correlation*) entre deux courbes, on fait « glisser » une courbe le long d'une autre courbe. En supposant les deux courbes de longueur égale, cela provoque que les points de l'extrémité de droite de la courbe glissée vers la droite ainsi que les points de l'extrémité de gauche de l'autre courbe sont alors comparées avec zéro.

Dans le cas courant, les courbes à comparer sont en fait les rayons d'un cercle plus ou moins déformé. On sait aussi que deux points représentant le même point réel auront le même *point fingerprint*, à une rotation près. En sachant cela, plutôt que d'effectuer une corrélation croisée standard, on effectue une corrélation croisée « circulaire » de façon à ce que les points de l'extrémité droite de la courbe glissée vers la droite soient replacés à l'extrémité gauche. Un exemple de cette situation est illustré à la figure 3.12.

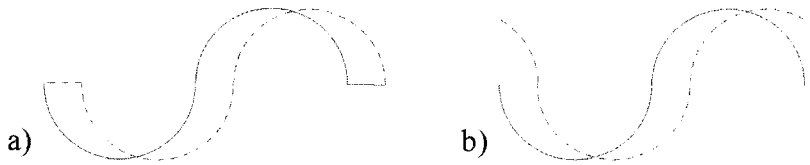


Figure 3.12: a) Corrélation croisée. b) Corrélation croisée circulaire

On voit bien à la figure 3.12a) la nécéssité d'ajouter une certaine quantité de zéros à chaque courbe pour pouvoir effetuer la corrélation croisée, ce qui augmenterait considérablement le nombre d'erreurs lors de l'appariement des points. Puisque les courbes à l'étude sont des cercles déformés, la configuration illustrée par la figure 3.12b) correspond mieux à nos besoins.

3.2.2.3 Projection 3D-2D

Pour obtenir des *point fingerprints*, il a été expliqué, au chapitre précédent, que les points 3D sont projetés sur le plan 2D possédant la même normale que le point. Ensuite, ces points sont rééchantillonnés selon K angles séparés de $2\pi/K$ chacun. Les auteurs de l'article ont cependant oublié de mentionner ce qui devait se passer pour le cas illustré à la figure 3.13.

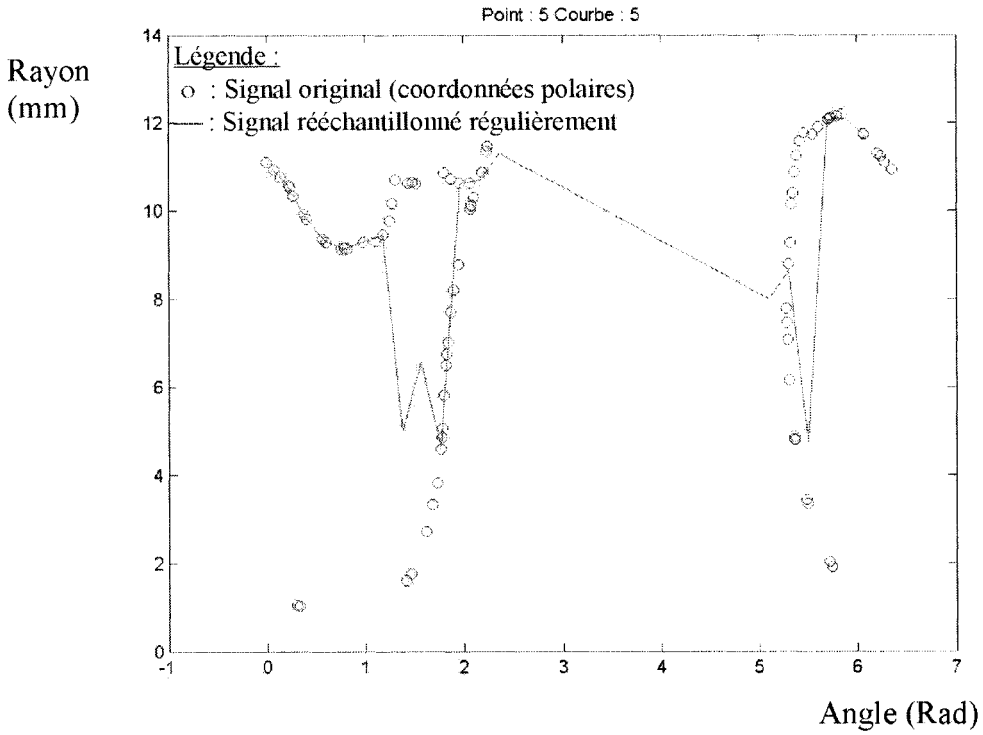
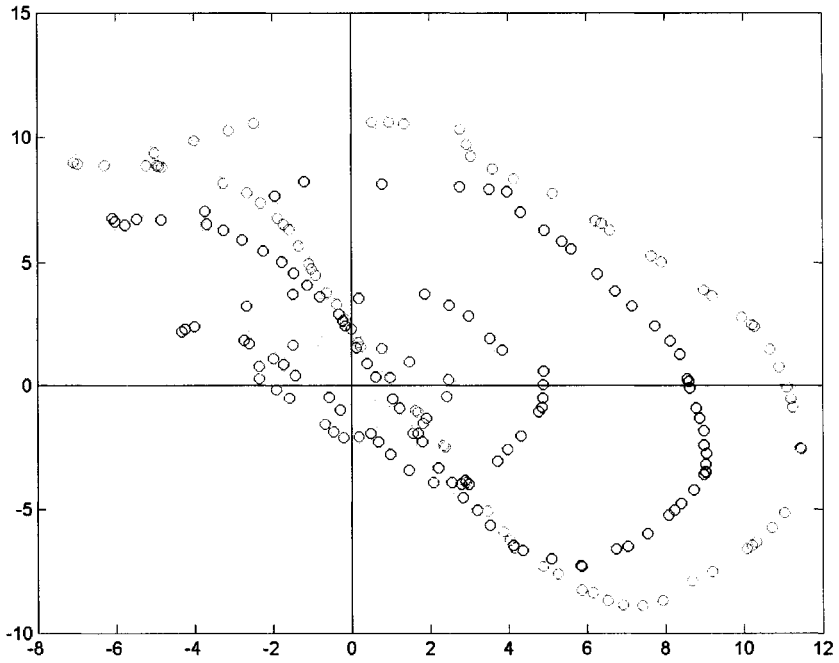


Figure 3.13: Dédoublement des courbes pour les mêmes angles

Ici, étrangement, il semble exister plusieurs courbes passant par les mêmes angles et aucun point n'est présent entre les angles $0{,}8\pi$ et $1{,}6\pi$ approximativement. Pour comprendre cette situation, il faut regarder la figure 3.14. On voit alors que la raison de ce phénomène est simplement le fait que la courbe est décentrée. À la figure 3.14a), on voit que le point se trouve dans une sorte de crevasse et à la figure 3.14b), on remarque pourquoi il existe de la confusion lors du rééchantillonnage de la 5^{e} courbe.

a)



b)

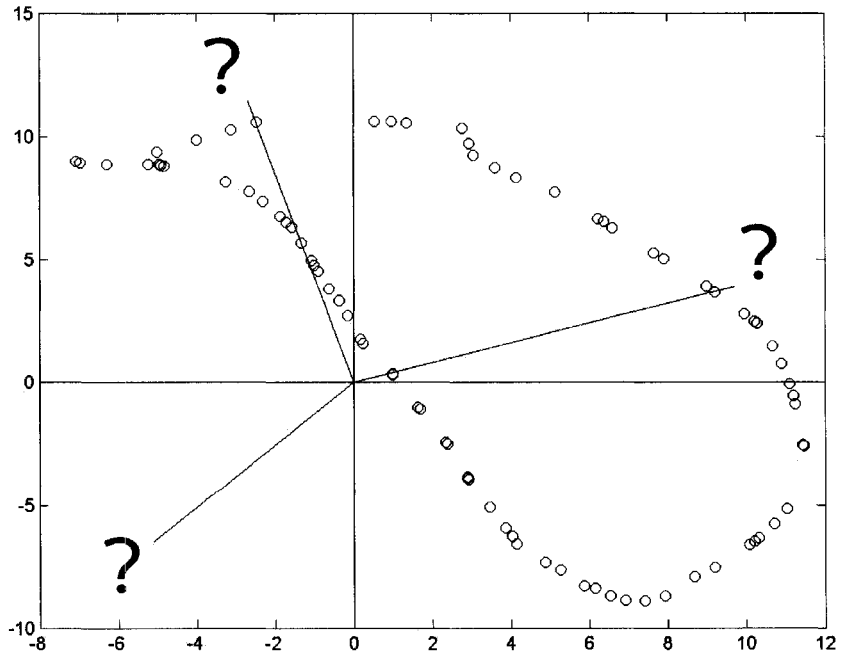


Figure 3.14: Courbe décentrée a) 5 courbes ensembles b) 5^{e} courbe

La situation illustrée ici est donc plausible et pose problème. L'appariement des points se fie en grande partie sur l'allure des courbes et le rééchantillonnage cause des zigzags puisque la courbe peut être rencontrée 2 fois pour les mêmes angles. De plus, le choix des *point fingerprints* se faisant sur le rapport des rayons maximum et minimum, une situation comme celle illustrée à la figure 3.14b) indique que si le décentrage de la courbe varie, le rapport des rayons change.

Pour pallier à ce problème, chaque courbe a été recentrée sur la moyenne de ses points avant d'être rééchantillonnée. De plus, la distance parcourue par la courbe pour se recentrer ou « décentrage » est conservée pour chaque courbe et pourra servir plus tard pour confirmer l'appariement des points.

3.2.2.4 Rééchantillonnage des courbes

Malgré le recentrage des courbes, il arrive parfois que le rééchantillonnage doive se faire sur une courbe possédant une certaine quantité de points situés approximativement aux mêmes angles, comme on peut le voir à la figure 3.15.

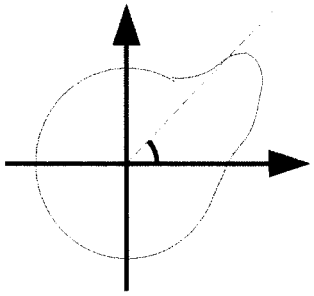


Figure 3.15: Courbe dont une partie est approximativement au même angle par rapport au centre de la courbe

Pour éviter de rééchantillonner la courbe avec deux points pris « au hasard » sur cette partie de courbe, le rééchantillonnage se fait plutôt en effectuant une moyenne des rayons entre les angles n et n+1. Ainsi, on évite de créer une fausse dissimilitude entre deux courbes représentant le même point sur deux surfaces différentes.

3.2.2.5 Vérification de l'appariement des points

Une fois les courbes correctement rééchantillonnées et appariées selon leur forme, il est possible d'ajouter quelques étapes simples qui permettront de vérifier si l'appariement des points est correct.

Tout d'abord, il est possible de vérifier que la différence entre les décentrages de chaque courbe est minimal. Il suffit simplement de calculer la somme des différences entre les décentrages pour des *points fingerprints* appariés. Les points appariés ayant une somme trop grande par rapport aux autres peuvent être facilement rejetés. La figure 3.16 montre que deux ensembles de courbes identiques peuvent représenter deux points ayant des reliefs très différents si les courbes ne sont pas centrées aux mêmes endroits.

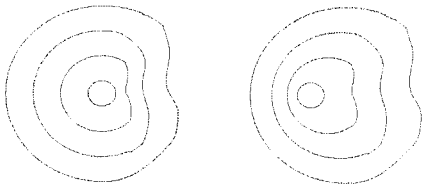


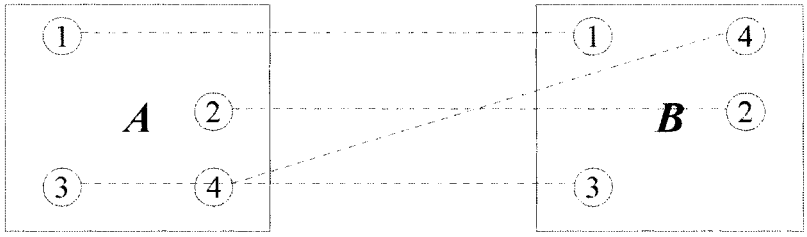
Figure 3.16: Ensemble de courbes semblables, mais centrées différemment

Ceci nous laisse un ensemble réduit de points appariés, mais il est possible que des erreurs se soient glissées parce que certains points non-correspondants semblent correspondre selon la forme et le décentrage de leur courbe. Il existe cependant assez d'information pour rendre une autre vérification possible. La méthode suivante consiste à effectuer une validation des appariements obtenus à l'étape précédente.

Supposons que nous avons les paires de points appariés (x_1,y_1) et (x_2,y_2), les points x_n appartenant à la première surface et les points y_n appartenant à la deuxième surface. Si les deux paires de points appariés sont valides, alors, logiquement, la distance entre x_1 et x_2 devrait être la même que la distance séparant y_1 et y_2.

Ainsi, pour n paires de points appariés, on crée un tableau n x n de différences entre les distances x_i à x_j et y_i à y_j. En supposant qu'à cette étape, plus de la moitié des paires de points sont valides, les paires de points possédant des différences de distances élevées par rapport à la moyenne pourront être éliminés.

La figure 3.17 présente un exemple concret de cette méthode. Supposons que les points 1 à 4 de la surface A ont été appariés avec les points 1 à 4 de la surface B. Le tableau présente le résultat des différences de distances dans un format binaire où '1' signifie que les distances sont identiques sur les deux surfaces et '0' signifie que les distances sont différentes (dans la pratique, puisque plus de la moitié appariements sont supposés corrects, les distances sous la moyenne sont considérées identiques et les distances au dessus de la moyenne sont considérées différentes). Par exemple, les distances A1-A2 et B1-B2 sont identiques, alors que les distances A1-A4 et B1-B4 sont différentes. Aussi, le tableau étant une matrice diagonale, il n'y a pas d'importance dans quel sens se fait la somme. On place donc les totaux en ordre, ce qui nous donne 2,1,3,4. On accepte automatiquement la paire A2-B2. Ensuite, on compare le nombre total de distances semblables au nombre de points déjà acceptés. Pour A1-B1, puisque le nombre total de points acceptés jusqu'à maintenant est plus petit ou égal au total obtenu (1 <= 2), on accepte A1-B1. Ainsi, pour A3-B3, le test (2 <= 2) est vrai, donc on accepte cette paire. Finalement, pour A4-B4, le test est (3 <= 1), ce qui donne faux. Le point est donc refusé. Les paires de points acceptables sont donc A1-B2, A2-B2 et A3-B3.



	A1-B1	A2-B2	A3-B3	A4-B4
A1-B1	-	1	1	0
A2-B2	1	-	1	1
A3-B3	1	1	-	0
A4-B4	0	1	0	-
Total	2	3	2	1

Figure 3.17: Ensemble de paires de points appariés

3.2.2.6 Optimisation de la performance

Calculer des distances géodésiques, effectuer des interpolations afin de rééchantillonner un signal et calculer des corrélations entre les points sont toutes des opérations qui peuvent être coûteuses en temps si elles sont répétées un grand nombre de fois, comme dans le cas courant. À cause de cela, il a fallu limiter les différents paramètres afin de permettre l'obtention de résultats satisfaisants dans un temps raisonnable.

D'après les recommandations fournies dans Sun 2003, les paramètres ont été fixés ainsi :

- 100 points d'intérêt détectés par surface
- 5 courbes par point (donc 5 rayons différents)
- 32 points par courbe (lors du rééchantillonnage)

Le choix des points d'intérêt ne s'effectuant que sur un rapport entre les rayons d'une même courbe, il est donc possible, et même souhaitable, de faire le choix des *point fingerprints* avant d'effectuer le rééchantillonnage. En effet, les interpolations entre les différents points formant les courbes peuvent adoucir certains maximums ou certains minimums et changer le résultat du rapport entre les rayons maximum et minimum, ce qui pourrait avoir comme conséquence de fausser le choix des points d'intérêt. De plus, cela permet de n'effectuer le rééchantillonnage des courbes que sur une toute petite fraction des points, ce qui permet de gagner du temps.

Finalement, pour éviter de choisir des points trop en bordure de la surface et pour éviter que certaines courbes ne permettent pas de discriminer les points d'intérêts les uns des autres, un minimum de trois points par courbe avant le rééchantillonnage est imposé pour qu'un point puisse être sélectionné comme point d'intérêt.

3.2.2.7 Simulation

Contrairement à la segmentation d'image où une foule d'images trouvées sur internet pouvaient servir de cas de test durant le développement de l'algorithme, les surfaces devant servir de cas de test sont ici générées dans MATLAB. Ceci permet de tester l'algorithme avant de se lancer dans le recalage de surfaces possédant plus de 30000 triangles.

3.2.2.7.a Surfaces synthétiques

La génération de surfaces synthétiques se fait de façon à ce que l'utilisateur puisse contrôler la rotation, la translation et la quantité de bruit blanc qui existe entre deux surfaces. La robustesse de chaque algorithme peut donc être évaluée malgré l'absence de surfaces acquises avec différents niveaux de bruit.

3.2.2.7.b Acquisition des surfaces réelles

Afin de procéder à la validation des méthodes, il a été possible d'utiliser des données réelles provenant d'acquisitions effectuées à l'Hôpital Ste-Justine grâce au système d'acquisition InSpeck.

De plus, les surfaces utilisées dans Sun 2003 sont directement disponibles sur le site de la base de données 3D de l'*Ohio State University*, Columbus :

http://sampl.eng.ohio-state.edu/~sampl/database.htm

ce qui a facilité la comparaison des résultats avec ceux de l'article. Les données contenues dans cette base de données sont fournies dans un format texte très facile à lire. Contrairement au VRML, par contre, ces données ne contenaient que les positions et l'ordre d'acquisition (ligne, colonne) des points, ce qui fait que les triangles ont dû être créés par la suite. Les données InSpeck et celles obtenues par internet ont environ le même nombre de points, mais les données internet ont davantage de relief (visage humain VS torse humain) et beaucoup moins de bruit.

3.2.2.7.c Implantation des algorithmes de recalage

Les algorithmes de recalage décrits précédemment ont été implantés afin d'en évaluer les performances sur des images réelles et/ou synthétiques. L'implantation des différentes méthodes a été faite en C++ et en faisant appel à certaines fonctions optimisées de MATLAB.

3.2.2.7.d Lecture des données d'InSpeck et logiciel de visualisation

Les données provenant des acquisitions peuvent être exportées dans différents formats. Cependant, la technologie InSpeck s'adressant en premier lieu au domaine de la modélisation 3D, la majorité des différents formats de données supportés sont donc les formats propriétaires de différents modeleurs 3D (par exemple, 3D Studio MAX ©). Un des formats non propriétaires supportés est le format VRML. La norme VRML étant complexe à implanter dans sa totalité, la modification d'un logiciel à code ouvert écrit par T. Garthwaite et J. Reposa a dû être effectuée afin de lire et d'écrire les fichiers VRML (gracieuseté de Jonathan Boisvert). Ainsi, ce logiciel a été adapté de façon à supporter les coordonnées de texture et afin d'ajouter le support des quadrilatères (la version originale du logiciel ne permettant que les surfaces à facettes triangulaires).

De plus, le support des textures à l'intérieur de MATLAB étant très limité (MATLAB ne permet l'utilisation de coordonnées de texture arbitraires ce qui est nécessaire lorsque les textures sont des images de projection), il a aussi fallu implanter un logiciel de visualisation 3D permettant de visualiser les surfaces recalées avec textures. Pour l'écriture de ce logiciel de visualisation, les fonctionnalités offertes par les bibliothèques de fonctions OpenGL et glut ont été utilisées (une fois de plus gracieuseté de Jonathan Boisvert). La plupart des résultats du chapitre suivant sont cependant présentés grâce aux outils de visualisation présents dans MATLAB.

3.3 Méthodes de validation

Possiblement une des étapes les plus importantes de toute recherche scientifique est l'élaboration d'une stratégie afin de fournir une validation non-biaisée des algorithmes développés pour la réalisation du projet.

Pour la détection des régions d'intérêt, puisque cette opération est pour l'instant effectuée manuellement, il sera possible de se servir des résultats obtenus par les opérateurs afin de valider les résultats de l'algorithme automatique. Ici, le problème qui se présente pour la validation est que les erreurs faites par l'opérateur et par la machine sont habituellement de nature assez différente. L'opérateur dessine habituellement un contour à l'aide d'un polygone ne contenant que quelques points, alors que l'algorithme, dans les meilleurs cas, pourrait très bien effectuer un contour beaucoup plus « exact » de la région d'intérêt. D'un autre côté, dans les pires cas, l'algorithme pourrait très bien ajouter un segment complet ne faisant pas partie du tout de la région d'intérêt. Pour tenter de compenser pour ce problème, une comparaison des contours sera effectuée pour le cas « inter-opérateurs » (ou « intra-opérateur », si les circonstances l'exigent) ainsi que pour le cas « opérateur vs algorithme » de segmentation. La comparaison sera faite en termes de pourcentage de correspondance entre les régions d'intérêt.

Pour le recalage des surfaces, il existe déjà un moyen d'effectuer le recalage, moyennant tout d'abord l'utilisation d'une toile placée dans le domaine numérisable. Encore une fois, les résultats obtenus via une opération manuelle pourront servir de référence pour l'étape de validation. Bien entendu, il ne sera possible d'effectuer une comparaison réellement quantitative qu'en déterminant la matrice de transformation entre les surfaces recalées manuellement ou automatiquement. Il est à noter qu'un bon recalage est aisément identifiable visuellement par le fait que les différentes surfaces s'entrecroisent fréquemment à l'endroit où les surfaces se superposent.

Chapitre 4 : Résultats et discussion

4.1 Choix des paramètres

Afin d'obtenir des algorithmes entièrement automatisés, il faut souvent fixer quelques paramètres pour que ceux-ci conviennent le mieux possible à la situation courante. Voici donc une courte explication par rapport à chacun de ces paramètres.

4.1.1 Paramètres de la segmentation automatique

Il n'y avait en fait qu'un seul paramètre à déterminer ici : le paramètre de ϕ du modèle de Potts. Rappelons que le paramètre ϕ indique la « force » de la tendance d'homogénéité imposée par le modèle de Potts.

La valeur de ϕ a été déterminée afin d'obtenir un PLIC qui maximise le plus souvent pour un nombre de segments K dans l'intervalle d'optimalité qualitative. Ensuite, les tests ont été effectués sur les images simples, puis les images médicales afin de revérifier l'ensemble des résultats. Ainsi, une nouvelle valeur de ϕ pouvait être estimée. Par ce processus itératif et qualitatif, la valeur $\phi = 10$ fut retenue.

Récapitulation de l'ensemble des paramètres utilisés :

- Nombre de segments minimum dans une image : 4
- Nombre de segments maximum dans une image : 10
- Nombre d'itération maximales de l'algorithme ICM : 10
- le paramètre de ϕ du modèle de Potts : 10
- Période d'échantillonnage : 4

Puisqu'une image contient normalement de 7 à 9 segments, l'utilisation de 4 à 10 segments fait du sens dans ce contexte. ICM converge habituellement en 5-6 itérations, alors limiter le nombre d'itérations à 10 sert simplement à éviter que l'algorithme effectue inutilement un nombre trop grand d'itérations s'il oscille entre deux configurations. Finalement, les images étant assez volumineuse, un sous-échantillonnage de l'image est effectué pour réduire le temps de calcul. L'utilisation d'une période d'échantillonnage de 4 permet de réduire une image 1024*768 (786432 pixels) en une image 256*192 (49152 pixels), ce qui est suffisant pour le calcul du PLIC.

4.1.2 Paramètres du recalage automatique

Pour l'algorithme PF, il faut déterminer la taille du rayon de chaque courbe, ce qui influence directement la distance géodésique maximale à calculer. Malheureusement, l'augmentation de cette distance augmente exponentiellement le temps de calcul. Pour limiter le temps de calcul, avec N rayons, on pose la distance géodésique maximale à calculer à (N + 1)*1.5*(précision de l'acquisition). La taille de chacun des N rayons est choisie comme étant de 1 à N fois environ 1.5 fois la précision de l'acquisition, ce qui est normalement une bonne approximation de la distance moyenne entre les points. Par exemple, pour 5 rayons séparés de 2.5mm, la distance maximale à calculer est de 15mm.

Récapitulation de l'ensemble des paramètres utilisés :

- Échelle pour les calculs : ~1.5*(précision de l'acquisition)
- Nombre de rayons par point fingerprint : 5
- Taille des rayons : [1, ..., 5]*Échelle
- Distance géodésique maximale : 6*Échelle
- Nombre de Points Par Courbe : 32
- Nombre de Point Fingerprints a obtenir : 100
- Pourcentage des points pour IPM : 80%

Il est à noter que tous les paramètres du recalage sont fixes à l'exception de l'échelle qui doit être fixée selon la surface à l'étude. La plupart de ces paramètres étaient suggérés dans Sun 2003 et se sont avérés satisfaisants. Aussi, la pourcentage des points pour IPM est utilisé pour l'élimination des correspondances lorsque les surfaces ne sont que partiellement correspondantes et ce paramètre n'a été utilise que pour les surfaces InSpeck.

4.2 Résultats de segmentation

L'algorithme de segmentation a été testé à la fois sur des images couleur RGB d'origine non médicale et sur des images obtenues des numériseurs pour lesquelles il existait une image de phase correspondante. Dans le contexte des images RGB, il n'y a pas de composante de phase et pas d'ombrage. Donc, les données RGB brutes furent traitées directement, sans prétraitement. Ces tests n'ont été effectués que pour valider l'algorithme de segmentation et ne sont pas essentiels à la méthodologie. Les résultats de ces segmentations ont tout de même été placées à l'annexe 3 pour le lecteur intéressé.

4.2.1 Essais sur des images de patientes scoliotiques

Les prochaines images présentées sont celles de patientes de l'Hôpital Sainte-Justine souffrant de la scoliose. Le suivi de ces patientes est effectué par topographie de surface.

Pour l'instant, une personne doit effectuer la segmentation de ces images manuellement avant de pouvoir procéder à la reconstruction 3D. Afin de produire une reconstruction, il faut avoir préalablement généré une image de phase à l'aide du logiciel FAPS 6.0 de InSpeck inc.

4.2.1.1 Comparaison inter-opérateurs

Afin d'effectuer la comparaison inter-opérateurs, 126 images de patients scoliotiques vus de dos ont été segmentées manuellement par deux opérateurs. Le premier opérateur est l'auteur de ce mémoire. Le second opérateur est Valérie Pazos, qui effectue régulièrement le détourage manuel des images à l'Hôpital Ste-Justine. Les deux opérateurs sont donc habitués à effectuer ce type d'opérations manuelles sur des images de phase. Nous avons effectués les détourages en même temps, chacun à notre poste. Nous avions convenu à l'avance de la façon de faire le détourage, ce qui pourrait avoir pour effet de biaiser les résultats. Malgré cela, les différences observées sont relativement grandes.

La façon de comparer les régions d'intérêt choisies par les deux opérateurs est d'effectuer une intersection des deux régions. Les pixels des régions communes (intersection) pour les deux opérateurs sont BON et les pixels des régions différentes (non-intersection) sont MAUVAIS. Le pourcentage de correspondance est alors calculé par le rapport des pixels BON sur la somme des pixels BON et MAUVAIS.

% Correspondance = pixels BON / (pixels BON + pixels MAUVAIS)

La figure 4.2 montre le résultat de cette comparaison. Malgré le fait que les opérateurs s'étaient donné des règles pour effectuer les détourages, les différences sont marquées. En moyenne, les surfaces correspondent à 76,03% et, au mieux, au alentours de 85% et au pire à environ 56%. On remarque tout de suite la nécessité d'un algorithme automatisé pour permettre d'avoir une certaine répétabilité dans les résultats.

Le problème est que le patient bouge presque toujours un peu pendant une acquisition, ce qui peut rendre le contour de celui-ci assez flou. Les opérateurs font ensuite de leur mieux pour interpréter ce contour. Ils doivent aussi choisir de leur mieux, et souvent arbitrairement, comment et où couper le cou, les bras et le bas du dos ne faisant pas partie de la région d'intérêt. Cette situation est illustrée à la figure 4.1 par un cas où les opérateurs avaient tout de même près de 83% de correspondance.

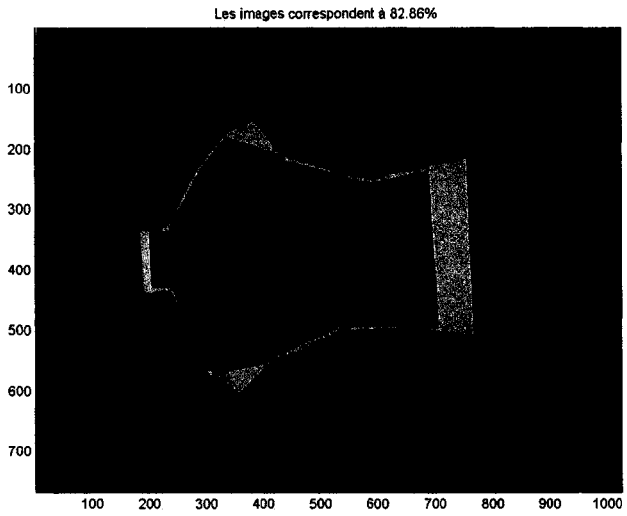


Figure 4.1: Exemple de comparaison inter-opérateurs

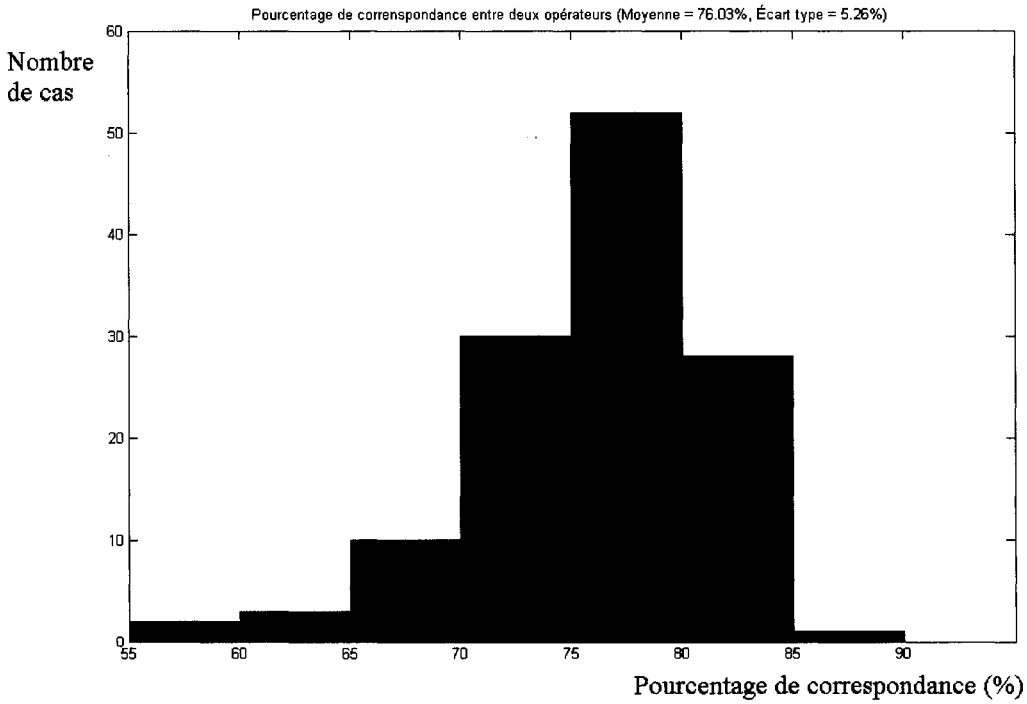


Figure 4.2: Comparaison inter-opérateurs

Sur la figure 4.1, la zone en rouge est la zone commune (intersection) aux deux opérateurs et la zone en vert est la zone différente pour les deux opérateurs.

4.2.1.2 Comparaison opérateurs-algorithme

Afin d'effectuer les comparaisons entre les résultats obtenus par les opérateurs et par l'algorithme de détection automatique de la région d'intérêt, un sous-échantillon des 126 images utilisées pour la comparaison inter-opérateur a été sélectionné. Parmi ces 126 images, 80 répondent à un certain standard du point de vue de la position et des vêtements, c'est-à-dire les bras élevés à environ 45 degrés du corps et pas de soutien-gorge. De ces 80, 22 possèdent une image de texture où les franges sont très visibles dû à un mouvement du patient durant l'acquisition, ce qui laisse 58 cas pour effectuer les comparaisons. Il est à noter que l'image de texture est reconstruite à partir des quatre images de franges, ce qui fait qu'un mouvement du patient peut faire apparaître des franges sur l'image de texture. Il est tout à fait possible que l'algorithme fonctionne aussi sur ces 22 images, mais il est préférable d'effectuer des tests sur des images qui ne sont pas inutilement bruitées.

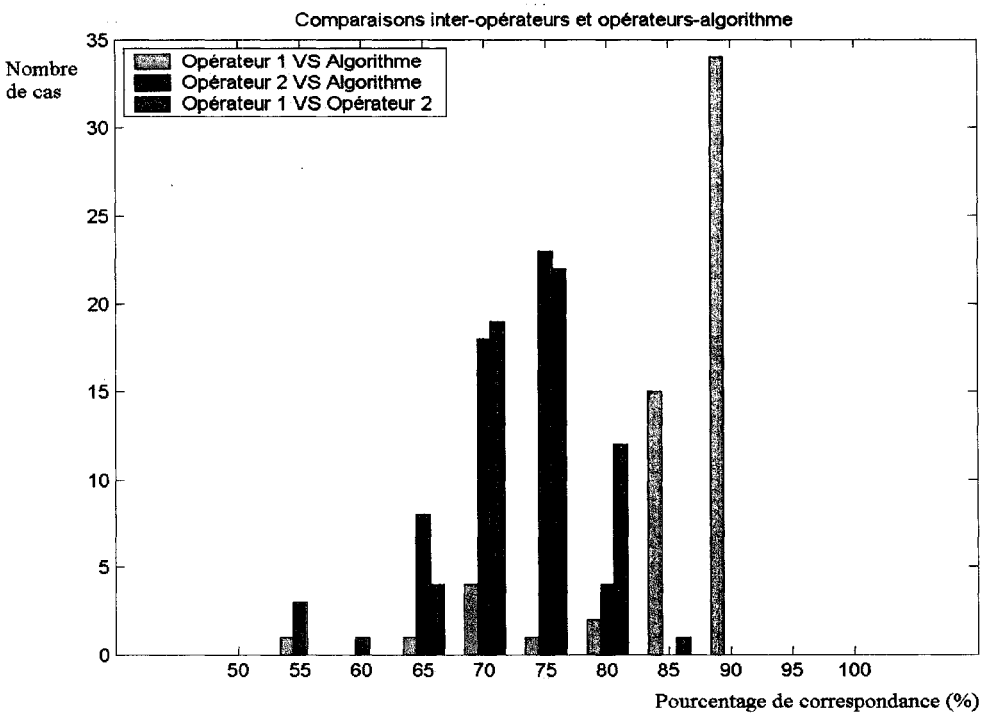


Figure 4.3: Comparaison inter-opérateurs et opérateurs-algorithme

La figure 4.3 nous montre la comparaison entre les différentes segmentations. L'histogramme répartit les données comprises dans plusieurs plages de données qui ont toutes 5% de large. Par exemple, les barres près du chiffre 55 indiquent le nombre d'occurrences de données comprises entre 55% et 60%. La comparaison inter-opérateur a été refaite ici avec seulement les 58 cas à l'étude. La similitude entre les segmentations de l'opérateur 1 et celles de l'algorithme est de loin plus grande que celle entre les segmentations des deux opérateurs. Les segmentations par l'algorithme ont été effectuées après celles effectuées par l'opérateur 1, donc celui-ci n'a pas pu s'inspirer des résultats pour biaiser la comparaison. Cependant, puisque l'opérateur 1 a implanté l'algorithme, il connaissait mieux les critères utilisés par l'algorithme pour choisir une région d'intérêt, ce qui explique les résultats obtenus. Le tableau 4.1 présente quelques résultats statistiques.

	Op 1 VS Algo	*Op 2 VS Algo*	*Op1 VS Op2*
Minimum	57,13%	49,09%	65,58%
Maximum	94,84%	82,87%	85,78%
Moyenne	88,34%	73,03%	75,98%
Écart-type	7,76%	6,45%	3,95%

Tableau 4.1: Statistiques comparatrices

À la figure 4.3, on remarque que 49 de 58 cas sont semblables à plus de 85% entre l'opérateur 1 et l'algorithme, alors que le maximum dépasse à peine 85% entre les deux opérateurs. Ceci démontre assez clairement que l'algorithme de détection de la région d'intérêt peut suivre certains critères aussi bien qu'un opérateur et que les décisions arbitraires prises par les opérateurs lors du détourage de la région d'intérêt peuvent s'éloigner de la solution recherchée.

4.2.1.3 Exemple de résultats

Comme mentionné au chapitre 2, on rappelle que, dans les images de phase, le passage brusque de la phase de 255 à 0 n'est pas nécessairement une discontinuité. Pour tenir compte de ceci, les algorithmes traitant des images de phase exécutent toujours un « déroulement de phase », c'est-à-dire qu'il est nécessaire de considérer que les pixels dont les valeurs sont faibles (foncés) peuvent être voisins des pixels ayant des valeurs élevées (pâles) sans qu'il y ait une véritable discontinuité dans le relief.

Dans les exemples de résultats fournis dans les pages suivantes, il faut prendre en considération que la composante de phase de ces images entre dans les calculs et que celles-ci sont utiles afin d'obtenir de l'information supplémentaire permettant de faire une meilleure segmentation. Cependant, ces images de phase ne seront pas affichées ici afin de simplifier l'apparence des tableaux de résultats, mais c'est en partie grâce aux images en phase que l'algorithme réussit à détacher de manière aussi efficace le corps (toutes les régions de peau exposées) de la patiente du reste de l'image.

Il est à noter que les contours présentés sont toujours des contours pleins (sans interruptions) et que les petites parties de contour parfois manquantes sont dues au zoom de l'image. On peut quand même bien voir le contour sur ces images.

Finalement, en regardant les résultats, il faut se rappeler que plus de 85% des résultats obtenus sont qualitativement bons et que les erreurs montrées sont presque les seules observées sur les 58 cas.

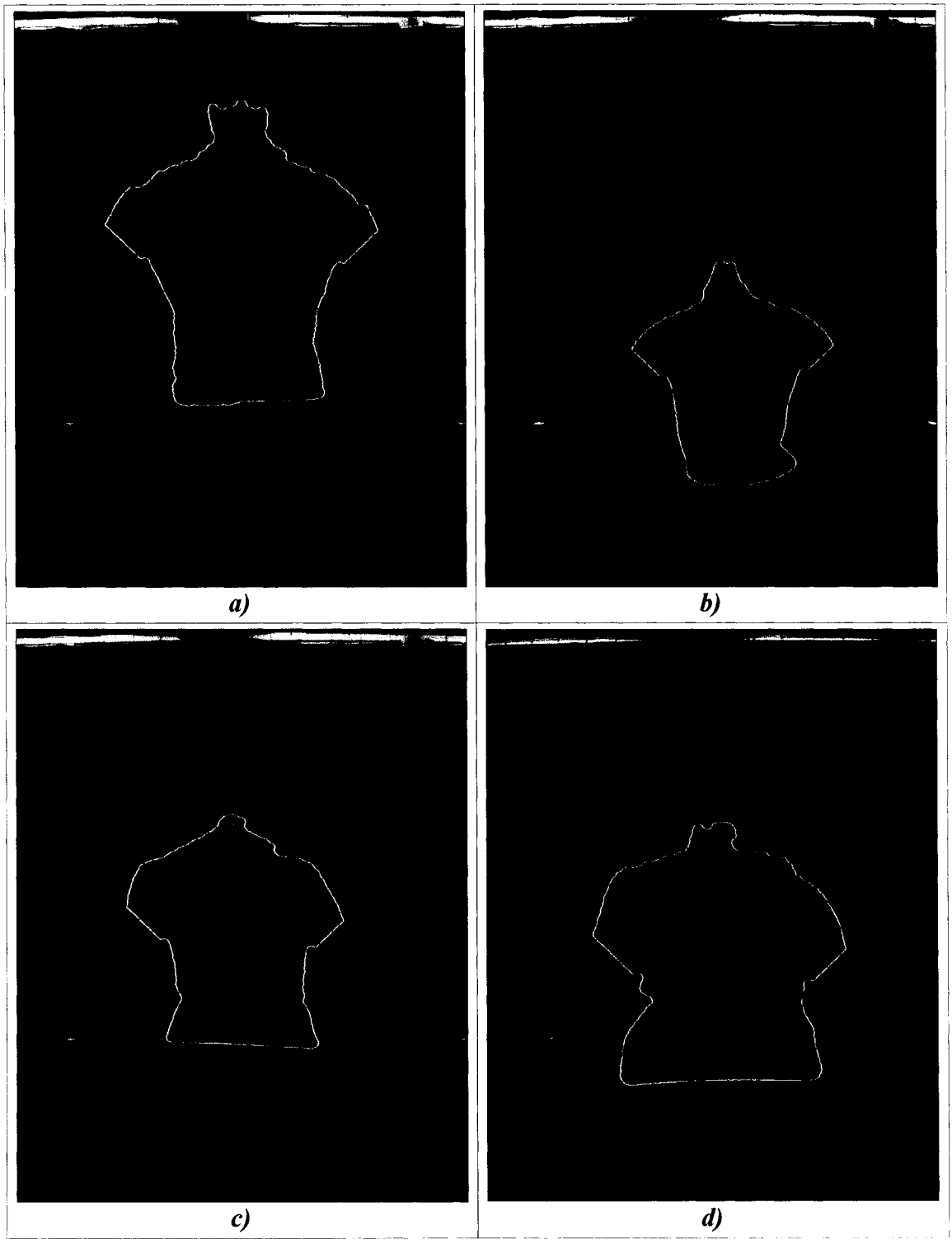


Figure 4.4: Exemples de bonnes segmentations

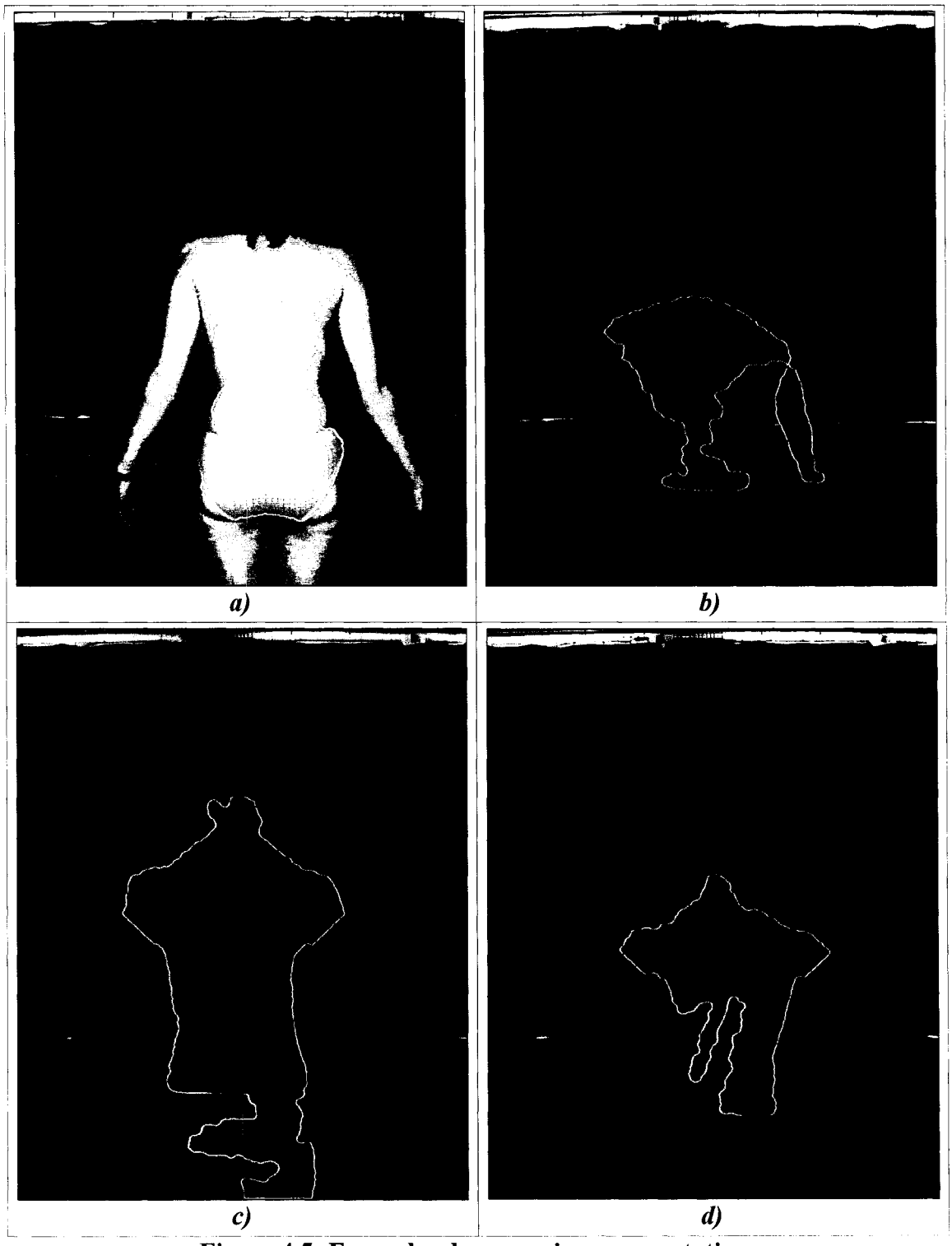


Figure 4.5: Exemples de mauvaises segmentations

À la figure 4.4, on voit bien que le fait que le sujet soit plus grand (en *a*), plus petit et plus mince (en *b*) ou plus corpulent (en *d*) n'est pas un problème pour l'algorithme.

À la figure 4.5, on peut voir quelques erreurs. En *a*, un éclairage excessif empêche une bonne détection des contours. Rappelons que selon le modèle de réflexion diffuse Lambertien, la lumière émise par une surface est modulée par la lumière incidente proportionnellement au cosinus de l'angle d'incidence. La transformation de RGB à YUV a tout de même ses limites, comme on le voit ici. Cette situation ne se produit cependant que sur cette image qui fait partie d'une série d'images plus ancienne que les autres et le problème d'éclairage semble avoir été réglé depuis ce temps.

En *b*, on observe le problème inverse. Une peau trop foncée et peu éclairée se confond aisément avec l'arrière plan. Ici, l'utilisation d'un écran bleu ou d'un écran vert comme arrière-plan aurait grandement aidé à la segmentation. C'est d'ailleurs la seule image où une des deux aisselles n'a pas été détectée correctement.

En *c*, on peut voir un cas rare où les sous-vêtements se sont joints au reste du corps. Des couleurs de sous-vêtements contrastant davantage avec la peau favoriserait aussi une meilleure segmentation. Il est cependant à noter que cette erreur peut être corrigée avec un minimum d'efforts en coupant la zone en ligne droite en suivant la culotte, ce qui donnerait tout de même un bon résultat presque reproductible d'un opérateur à l'autre vu la très faible différence inter-opérateurs qu'il peut y avoir à tracer une ligne droite. En supposant que cette situation ne se reproduise que très rarement, cela fait partie de l'erreur « acceptable » d'un algorithme entièrement automatisé.

Finalement, en *d*, la segmentation a raté parce que les tons de la peau semblaient changer brusquement. Encore une fois ici, l'utilisation d'une toile de fond (écran bleu ou écran vert) aurait pu permettre au PLIC de minimiser plus rapidement et ainsi de subdiviser l'image en moins de segments, ce qui aurait fourni une meilleure segmentation finale.

4.2.2 Discussion sur la segmentation d'images

Les algorithmes ICM et PLIC ont été programmés avec succès et les résultats obtenus ont été, dans la plupart des cas, satisfaisants. Comme mentionné dans la section « méthodologie », plusieurs points du PLIC ont été modifiés par rapport à la forme initiale suggérée dans l'article de Stanford et Raftery afin de simplifier le problème et d'obtenir de meilleurs résultats dans le contexte de la topographie de surface. Cette adaptation s'est avérée fructueuse dans les résultats. Aussi, la détection des aisselles et la coupure des bras à 45 degrés semble fonctionner parfaitement, tant et aussi longtemps que des bras sont présents dans la région d'intérêt.

ICM est un algorithme suffisamment efficace pour pouvoir l'utiliser à l'intérieur de nos itérations pour le PLIC sans que cela ne cause trop de délais pour l'utilisateur, en particulier si on le compare avec le recuit simulé (Kirkpatrick 1983). Cependant, ICM produit une classification sous optimale alors que le recuit simulé fournit une classification optimale. Visuellement, ICM est souvent satisfaisant, ce qui fait qu'autant que possible, il sera plus efficace de créer des conditions adéquates pour la capture des images plutôt que d'ajouter au temps de calcul de la segmentation automatique.

Même si le résultat optimal du PLIC est difficile à déterminer, les résultats obtenus par le PLIC sont souvent optimaux ou près du résultat optimal, dans le sens où le résultat optimal est considéré comme étant le plus petit nombre de classes permettant une classification correcte de l'objet ou de la personne à détacher du reste de l'image. La plupart des images obtenaient expérimentalement une segmentation optimale avec 7 à 9 segments distincts, ce qui semble souvent approprié. Cette constatation est encourageante pour les usages futurs possibles de cette méthode et il est probable que le PLIC se retrouve parmi plusieurs projets futurs en matière de segmentation automatique d'images.

La combinaison des images de phases et des composantes U et V des images converties en format YUV semble la plupart du temps suffisante pour obtenir tous les renseignements nécessaires à une segmentation automatique réussie. Aussi, les critères de sélection automatique de la région d'intérêt sont à la fois simples et efficaces et ont un très faible taux d'échec.

Idéalement, il serait bien de mettre un écran monochromatique en arrière-plan dans chacun des angles de caméra (pour tous les numériseurs).

4.2.2.1 Description de la portée des résultats obtenus

Les résultats obtenus sont qualitativement acceptables dans la plupart des cas, à quelques exceptions près. Les images floues sont difficiles à segmenter, comme prévu, et les images ayant beaucoup de sections distinctes n'obtiennent pas toujours un nombre de segments (déterminé par le PLIC) suffisamment grand pour permettre de distinguer chaque section. D'après ce qui a pu être observé au cours des divers tests, la méthode est prometteuse pour ce qui est de l'utilisation en clinique.

Pour ce qui est de la performance de la méthode, le temps du calcul effectué avec un pentium 4, 2 GHz et 512 Mb de mémoire RAM était, expérimentalement, d'environ trois à quatre minutes. Ce temps inclut toutes les étapes, dont le calcul du PLIC optimal incluant lui-même plusieurs itérations de ICM à chaque itération du PLIC, l'affichage des résultats à chaque étape, le calcul du ICM final sur l'image complète et les opérations *open/close* effectuées avec les fonctions *imopen* et *imclose* de MATLAB. Aussi, ce temps inclus au moins une minute de conversion du contour en format InSpeck (fichier avec extension .iaf), de lecture et d'écriture de fichiers.

Il est à noter qu'avant l'utilisation de la segmentation EM pour initialiser l'algorithme, des algorithmes pseudo-aléatoires, tel le *fuzzy C-means clustering*, ont été utilisés dans les premières versions de l'algorithme de segmentation, mais cela fournissait occasionnellement des résultats visiblement moins bons, donc non-reproductibles.

Le programme est entièrement écrit en MATLAB, alors en supposant l'utilisation de cette méthode dans un programme écrit en C++ dans lequel les images seraient déjà chargées en mémoire, moins d'une minute devrait suffire pour obtenir la segmentation finale. Cette diminution potentielle du temps d'exécution s'explique par le fait que MATLAB est un langage interprété efficace pour les opérations matricielles, mais inefficace pour les boucles. Les algorithmes récursifs, comme l'algorithme courant, s'exécutent habituellement beaucoup plus rapidement lorsqu'un langage compilé, comme le C++, est utilisé. Lors de la conception, le choix s'est arrêté sur MATLAB simplement parce que la phase d'implémentation est moins longue en MATLAB qu'en C++.

Dans notre cas, l'intérêt est de détecter le torse d'un patient atteint de la scoliose, mais, en choisissant différents critères pour le choix des régions, cette méthode pourrait très bien être utilisée pour détecter des tumeurs sur un PET Scan (Tomographie d'Émission par Positrons (TEP)) ou, dans un tout autre domaine, pour détecter des champs de marijuana sur une photo satellite. L'algorithme ICM est déjà utilisé dans une multitude d'applications de la sorte, mais le PLIC permettrait de généraliser la technique de résolution et de réduire le nombre de données nécessaires que l'utilisateur doit connaître *a priori* afin d'utiliser ICM.

4.3 Résultats de recalage des surfaces

Pour valider les algorithmes IPM et PF, des images synthétiques et des images réelles ont été utilisées afin de vérifier à la fois l'exactitude, la robustesse et la performance de l'implémentation réalisée.

L'idée derrière l'utilisation de plusieurs types d'images est de procéder à la validation de l'algorithme dans des conditions contrôlées et de continuer la validation avec un environnement de moins en moins contrôlé afin d'augmenter progressivement le niveau de difficulté. Voici l'ordre des images utilisées :

1) Surfaces synthétiques (Contrôle de la forme des surfaces et du bruit);
2) Surfaces réelles sans le problème d'illumination (Bruit faible);
3) Surfaces InSpeck.

Comme nous le verrons plus tard, les surfaces InSpeck contiennent un bruit assez fort dû au mouvement du patient durant la projection des franges (problème d'illumination) et le pourcentage de correspondance entre deux surfaces est aussi plus faible pour les surface InSpeck que pour les autres types de surfaces, ce qui fait que le recalage des surfaces InSpeck est beaucoup plus difficile.

4.3.1 Tests sur des surfaces synthétiques

Pour débuter, voici quelques tests montrant des résultats de la méthode IPM. Chaque cas de départ est une rotation de $\pi/8$ selon un axe de coordonnées (X, Y ou Z). Les données sont particulièrement difficiles à recaler par IPM puique les deux surfaces ont très peu de relief. La figure 4.6 montre que l'algorithme réussit habituellement à recaler assez bien les surfaces pour les rotations selon tous les axes. Les rotations coplanaires entre deux surfaces, comme dans le 3e cas, sont toujours les plus difficiles à recaler parce qu'il est plus facile de tomber dans un minimum local et le recalage est en effet un peu moins bon. Cette situation sera discutée un peu plus tard.

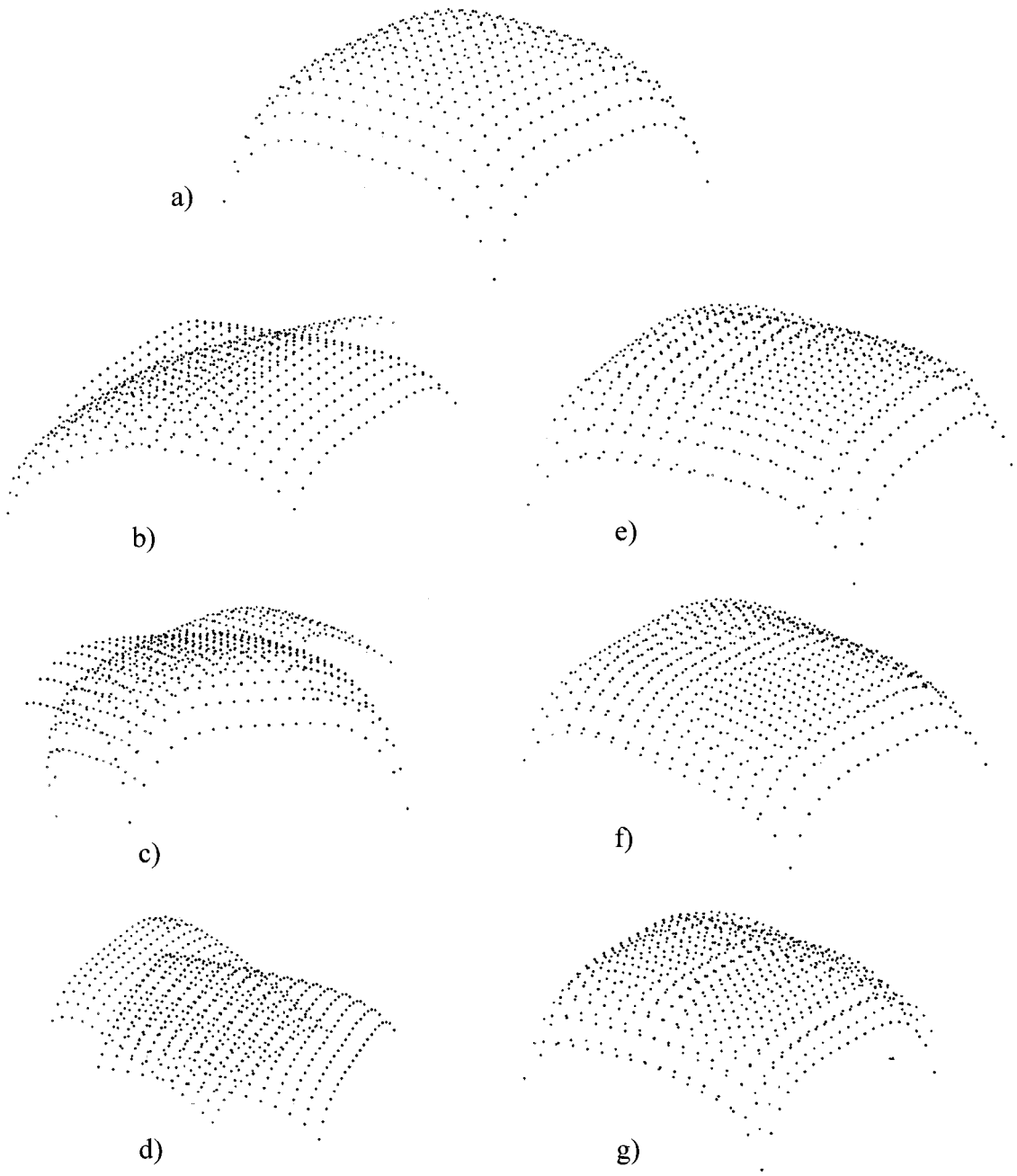


Figure 4.6: Résultats obtenus par IPM. En a : Position idéale. En b, c et d : Ajout d'une rotation. En e, f et g : Résultat du recalage de l'image de gauche par IPM.

Puisqu'il est difficile pour IPM de recaler des objets ayant une forme similaire à un plan, les tests suivants montrent le recalage d'un plan bruité par la méthode PF suivi du raffinement par IPM. Les plans sont générés avec des valeurs fixes en X et en Y pour former une grille dont les branches sont de longueur unitaire et le relief est créé par un bruit blanc d'amplitude 1 (Le même bruit est utilisé pour les deux surfaces). Le bruit qui différencie les deux surfaces est aussi un bruit blanc et est ajouté au bruit blanc existant. Donc, par exemple, le bruit de 50% est un bruit blanc d'amplitude 0,50 ajouté au bruit blanc d'amplitude 1 pour la seconde surface. Dans ces conditions, la méthode PF trouve des points d'intérêts permettant un bon recalage jusqu'à environ 50% de bruit.

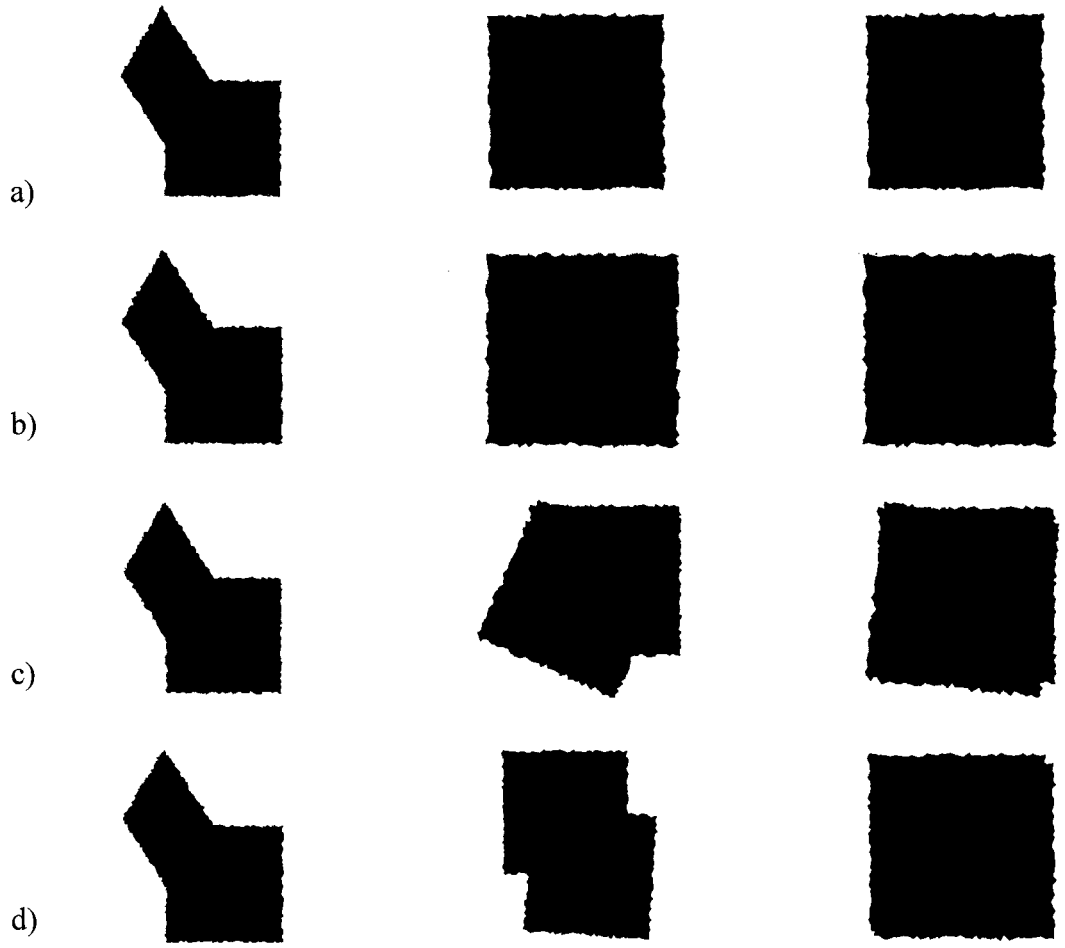


Figure 4.7: Recalage de plans. Les bruits sont de : a) 1%, b) 50%, c) 66%, d) 75%

4.3.2 Tests sur des surfaces réelles

Puisque les visages utilisés dans l'article de Sun 2003 sont disponibles sur internet, ils seront utilisés pour valider la méthode. En second lieu, des images provenant d'acquisitions InSpeck seront utilisés pour vérifier la robustesse de l'ensemble de la méthode.

4.3.2.1 Images de visages humains

Les visages ont été téléchargés à partir de la page internet suivante :

http://sampl.eng.ohio-state.edu/~sampl/data/3DDB/RID/minolta/faceimages.0300/index.html

Les résultats obtenus utilisent la méthode PF avec les modification mentionnées dans la méthodologie suivie par un raffinement par IPM. Tous les recalages ont été effectués avec les mêmes paramètres (5 rayons séparés de 2.5mm).

Les figures 4.8 et 4.9 montrent deux résultats qui ont réussit avec et sans zone d'exclusion. Rappelons que la zone d'exclusion permet de définir une zone autour d'un point d'intérêt sélectionné dans laquelle aucun autre point d'intérêt ne peut être sélectionné. Les figures 4.10 et 4.11 montrent aussi deux résultats avec et sans zone d'exclusion, mais ici, le cas sans zone d'exclusion échoue. La figure 4.12 montre que, sans zone d'exclusion, tous les points d'intérêt du visage se concentrent, pour cette image, sous le nez et dans les oreilles, alors qu'avec la zone d'exclusion, les points se distribuent un peu plus dans la zone des yeux. La raison pour laquelle le visage se retrouve recalé à l'envers (le front vis-à-vis le menton) est que, lorsque les points sont concentrés sous le nez et les oreilles, une image trouve des points dans l'oreille gauche et ceux-ci sont appariés avec les points de l'oreille droite de l'autre image. Puisque la distance entre le nez et l'une ou l'autre des oreilles est sensiblement la même, les points d'intérêt passent avec succès tous les critères de sélection et un recalage erroné est obtenu. Il est à remarquer que l'algorithme IPM effectue un très bon travail pour raffiner le recalage puisque les figures 4.8 et 4.9 obtiennent pratiquement le même recalage et, malgré une approximation plus ou moins juste de la part de la méthode PF à la figure 4.10, le recalage obtenu est excellent.

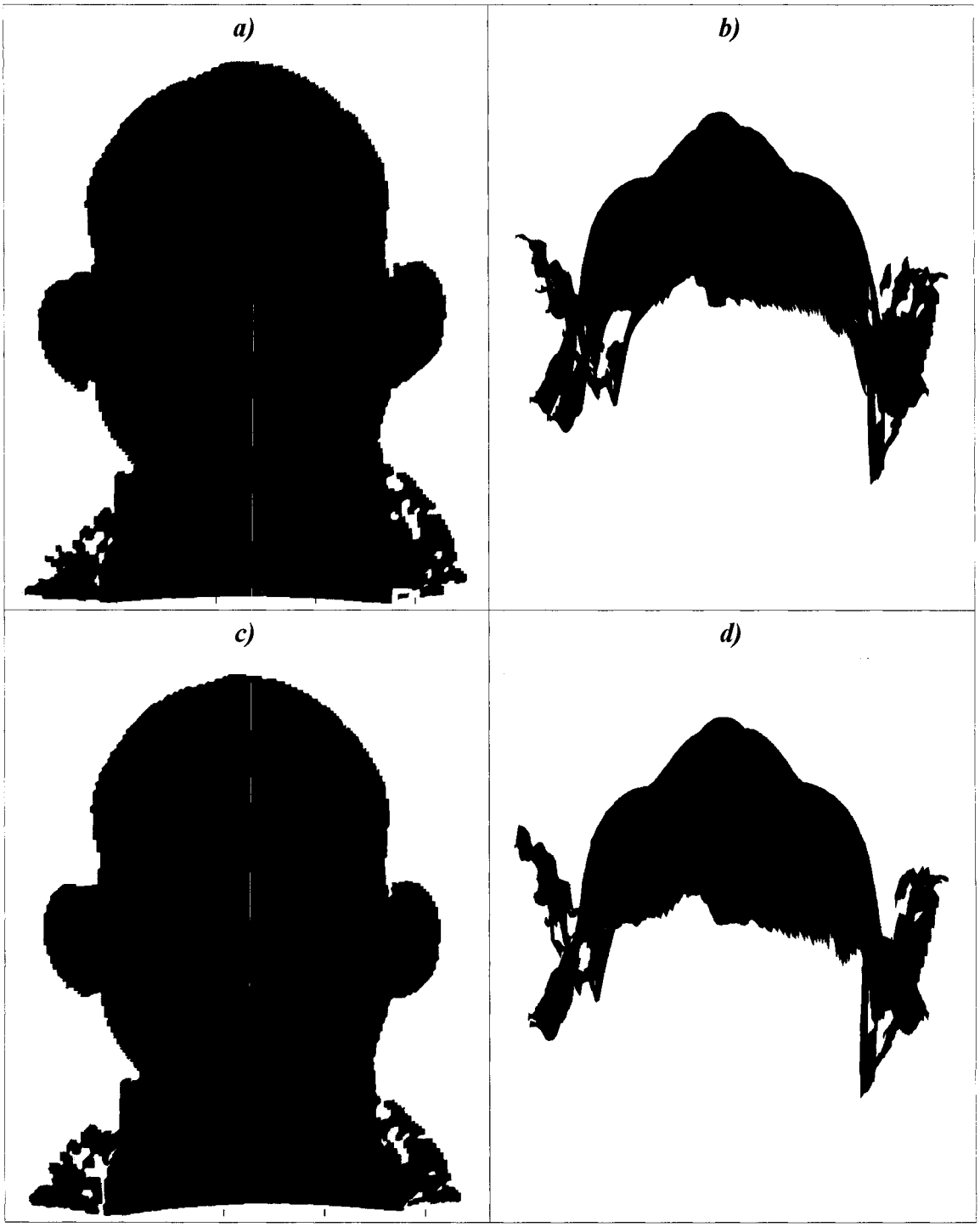


Figure 4.8: Recalage avec zone d'exclusion. a) et b) Recalage par la méthode PF. c) et d) Raffinement par IPM

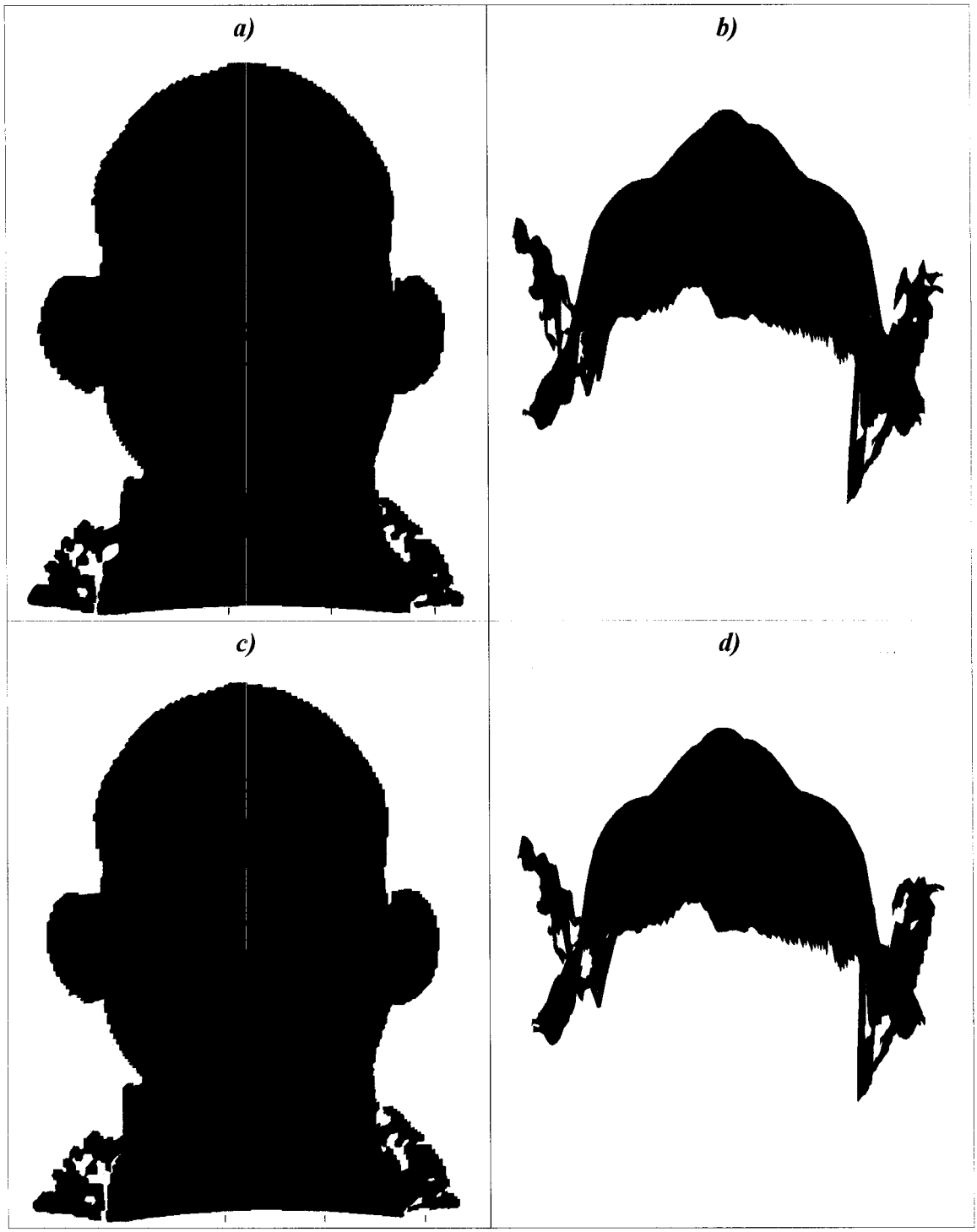


Figure 4.9: Recalage sans zone d'exclusion. a) et b) Recalage par la méthode PF. c) et d) Raffinement par IPM

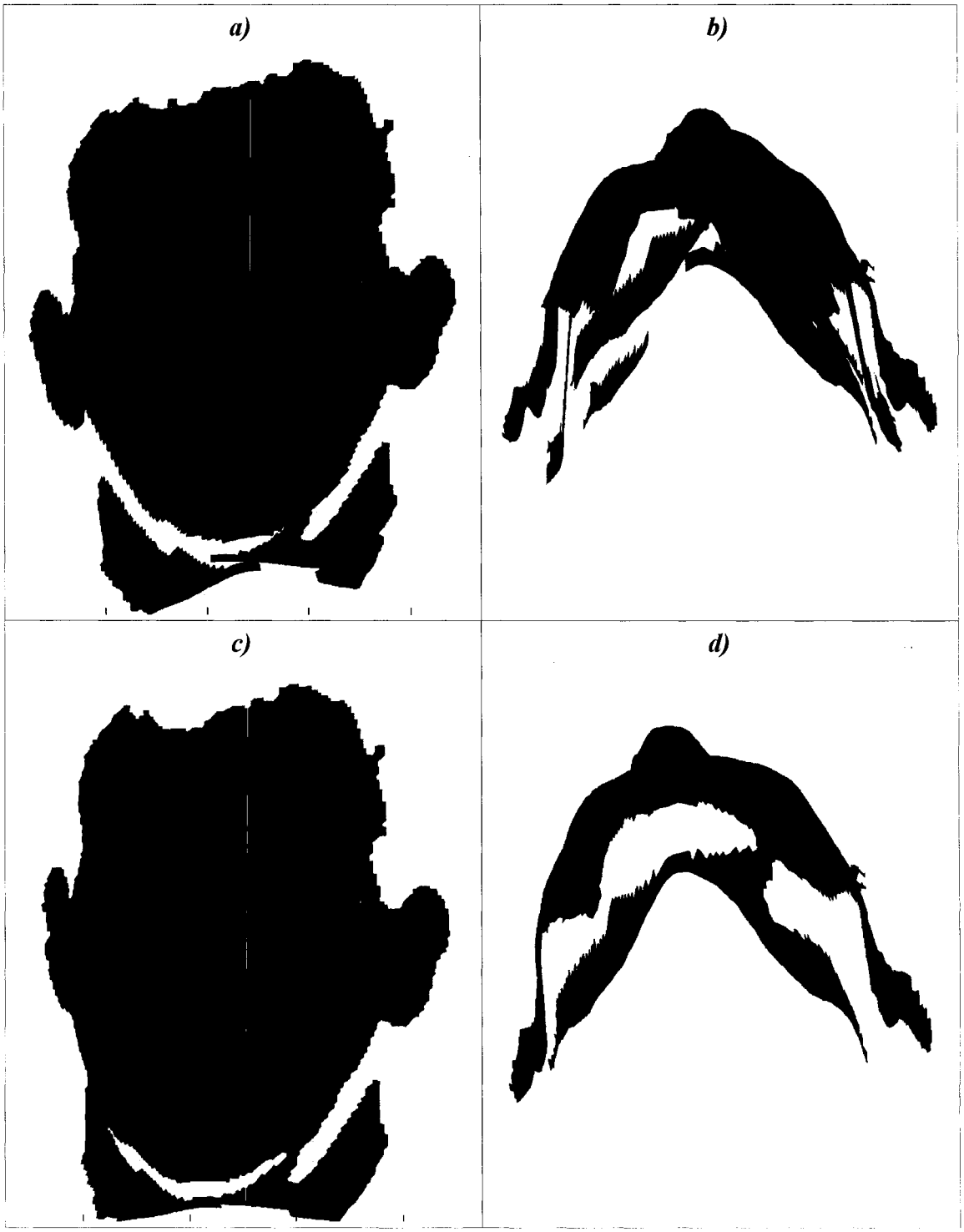


Figure 4.10: Recalage avec zone d'exclusion. a) et b) Recalage par la méthode PF. c) et d) Raffinement par IPM

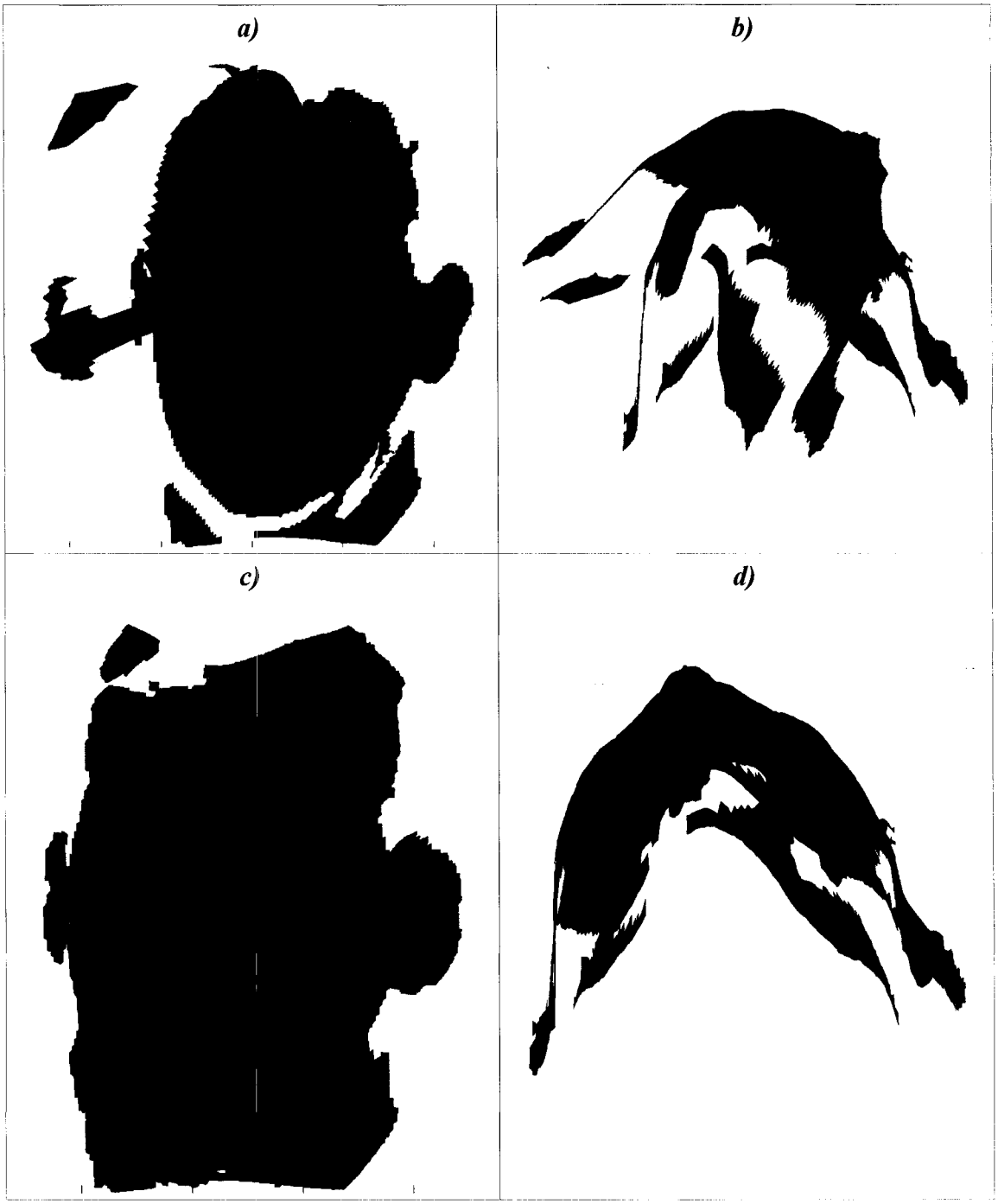


Figure 4.11: Recalage sans zone d'exclusion. a) et b) Recalage par la méthode PF. c) et d) Raffinement par IPM

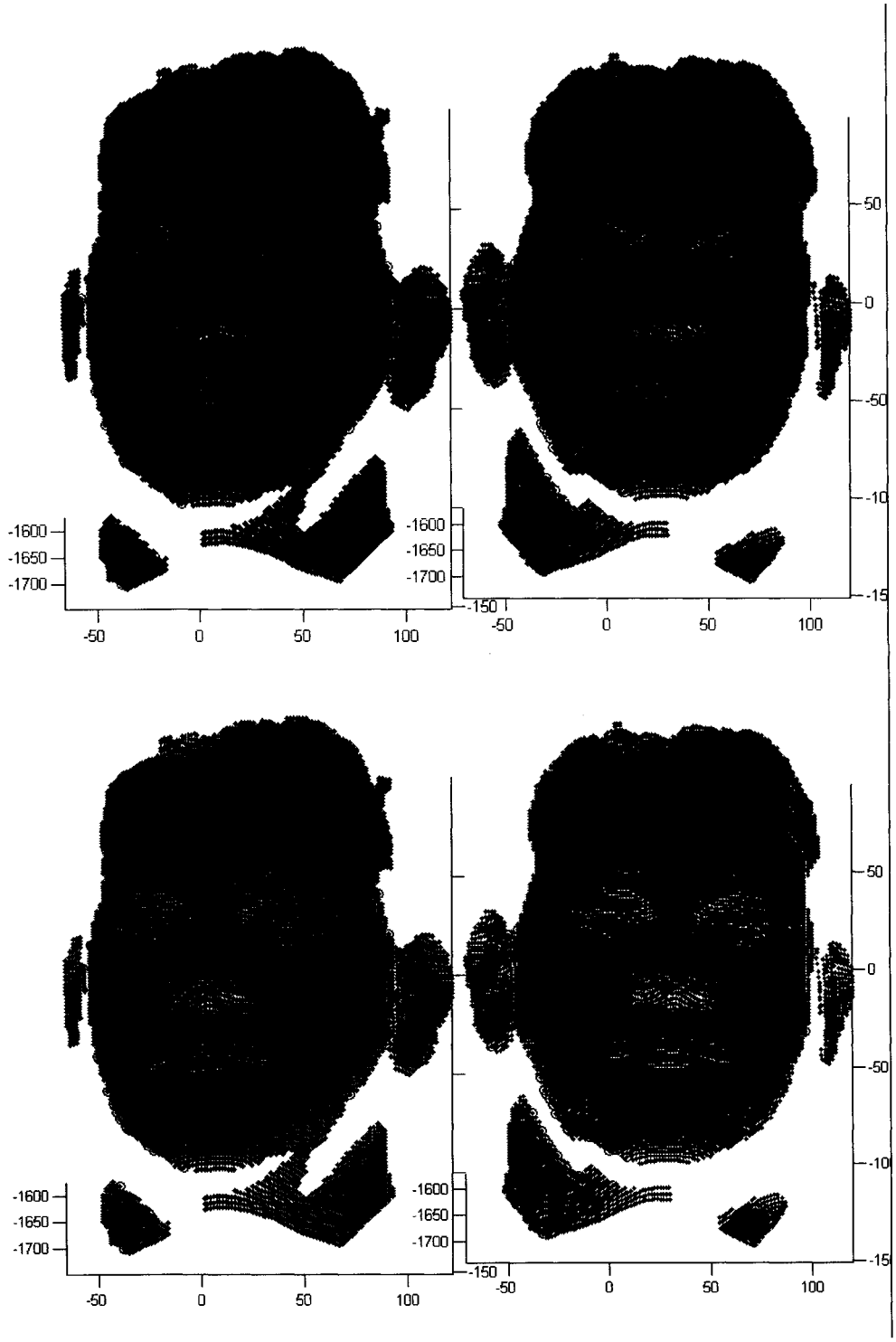


Figure 4.12: Points d'intérêt. Haut : Sans zone d'exclusion. Bas : Avec zone d'exclusion

4.3.2.2 Images InSpeck

Une différence importante entre les images obtenues par la méthode InSpeck et les autres images utilisées précédemment est que ces surfaces contiennent des parties correspondantes moins grandes en proportion que les surfaces montrées jusqu'à maintenant. Par exemple, alors que les surfaces synthétiques et les surfaces de visages humains se superposent à au moins 95%, les surfaces InSpeck se superposent à environ 80%, ce qui veut dire que 80% d'une surface peut trouver des correspondances sur l'autre surface.

Aussi, il est de toute évidence plus difficile de trouver des points d'intérêt sur un torse que sur un visage et certains attributs, comme les seins, peuvent fournir des zones propices à la détection de points d'intérêt pour certaines filles, mais cette zone fournit habituellement peu d'information utile au recalage chez les garçons. De plus, de par leur nature, les topographies de surfaces sont souvent plus bruitées en bordure, ce qui a tendance à y attirer un trop grand nombre de points d'intérêt. Pour ces raisons, la zone d'exclusion devient ici importante pour essayer autant que possible de répartir les points d'intérêt un peu mieux sur toute la surface et ainsi espérer que l'appariement des points correspondants soit possible.

Ensuite, il est clair que l'algorithme IPM ne pourra être utilisé tel quel. Si on essaie de minimiser les distances entre tous les points de deux surfaces partiellement correspondantes, le recalage sera raté. La figure 4.6 montrait d'ailleurs que IPM fournit un minimum local pour tous les points, ce qui a davantage tendance à superposer les deux surfaces le plus possible plutôt que de superposer les bonnes parties des surfaces seulement. Pour cette raison, un certain pourcentage des points seulement pourra être utilisé pour recaler les deux surfaces par IPM. Si la méthode PF fournit un bon recalage de départ, les points d'une surface A les plus près des points d'une surface B pourront servir à effectuer le recalage, comme expliqué plus tôt dans la section « élimination de correspondances ».

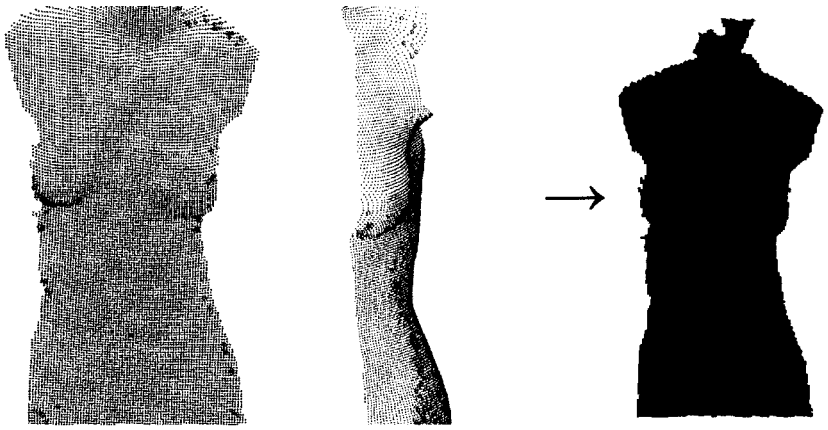


Figure 4.13: Recalage par IPM de deux surfaces minimalement correspondantes

À la figure 4.13, on voit qu'en utilisant IPM pour recaler deux surfaces à peine correspondantes, le minium trouvé ramène la seconde surface vers le centre de la première surface. Ce genre de surfaces contient aussi trop peu de points correspondants pour que la méthode PF fonctionne correctement. Il est donc important de maximiser la zone commune aux deux surfaces à recaler. Les surfaces ont donc dû être générées avec cette idée en tête.

Pour cette raison, 26 ensembles de 3 images (images avant, avant-gauche et avant-droit de patients) ont été détourées en prenant soin de maximiser la zone commune aux deux images. De ces images ont été reconstruits 26 ensembles de 3 surfaces.

Malgré cela, l'algorithme PF n'est pratiquement d'aucune utilité sur ces images. Cela s'explique par le fait que cet algorithme utilise les déformations locales pour effectuer l'appariement et que les surfaces acquises contiennent du bruit important causé par le mouvement du patient durant l'acquisition. Ce mouvement durant la projection de franges cause un flou de mouvement qui est clairement visible sous la forme d'un bruit sinusoïdal vertical à la figure 4.14. Ce bruit affecte beaucoup trop les normales des points de la surface pour que la méthode PF puisse fonctionner dans ce contexte.

Le problème important causé par ce bruit est que le filtrage de ces images par un filtre qui éliminerait le bruit éliminerait aussi beaucoup trop de déformations locales, ce qui rendrait inutile l'algorithme PF.

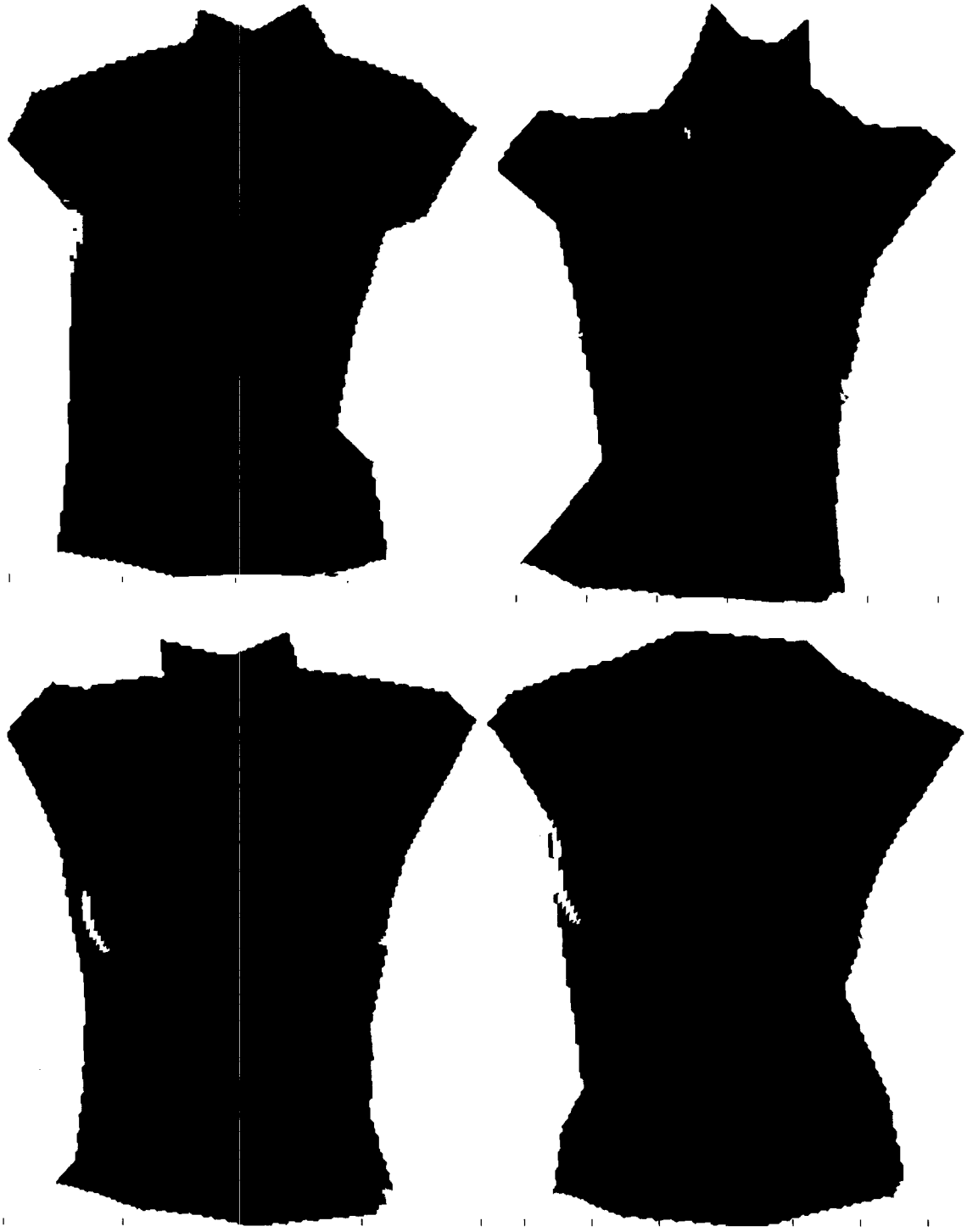


Figure 4.14: Exemples de torses provenant d'acquisitions InSpeck

À cause de cela, il faudrait développer un filtre qui élimine spécifiquement ce type de bruit en conservant un maximum d'informations sur le torse. Aussi, plus le temps d'acquisition de l'appareil sera court, plus le flou de mouvement sera réduit.

Malgré cela, quelques tests ont été faits afin de voir comment réagit IPM à un mauvais recalage de départ. Pour ce faire, les images avant et avant-droites des patients ont été recalées par la méthode PF, puis par IPM. Plusieurs problèmes causés par le bruit sinusoïdal sont alors apparus. Tout d'abord, habituellement, au moins une des deux surfaces est visiblement bruitée, ce qui réduit les chances d'un appariement correct. Ensuite, le bruit ajoute des points d'intérêt là où il n'y en a pas. Enfin, le bruit est presque toujours de phase différente et de fréquence différente sur les deux images, ce qui élimine les points d'intérêt valides. À la figure 4.15, un exemple de recalage réussi est montré. Le fait que le recalage par la méthode PF (figure 4.15a) ne soit pas trop mauvais est en partie le fruit du hasard. Pour que la vérification des distances entre les points appariés élimine toutes les paires de points non valides, il faut qu'au moins la moitié des paires de points appariés par la méthode PF soient valides, ce qui n'est presque jamais le cas ici à cause du bruit. Les paires de points appariés ici ne sont pas bonnes, mais sont au moins proches. Une fois ce recalage obtenu, 80% des points de la surface avant-droite ont été conservés pour le recalage par IPM. Ces points sont en rouge sur la figure 4.15b. La valeur « 80% » a été choisie approximativement à l'oeil en supposant que, de façon générale, environ 80% des points de la surface avant-droite se retrouvent sur la surface avant. Cette valeur aurait aussi pu être déduite à partir de la configuration géométrique des caméras. Pour choisir les points, on calcule les distances de tous les points d'une surface par rapport aux points de l'autre surface et on prend simplement les 80% les plus près. Les figures 4.15c et 4.15d montrent le résultat de ce recalage, qui est assez bon, puisqu'on voit les couleurs rouge et bleu s'entrecroiser. On remarque aussi la présence du bruit grâce à cet entrecroisement de couleurs. La figure 4.16 présente un cas raté. On voit que si le recalage de départ est trop mauvais, les mauvais points sont choisis pour effectuer IPM, ce qui conduit à un mauvais recalage final.

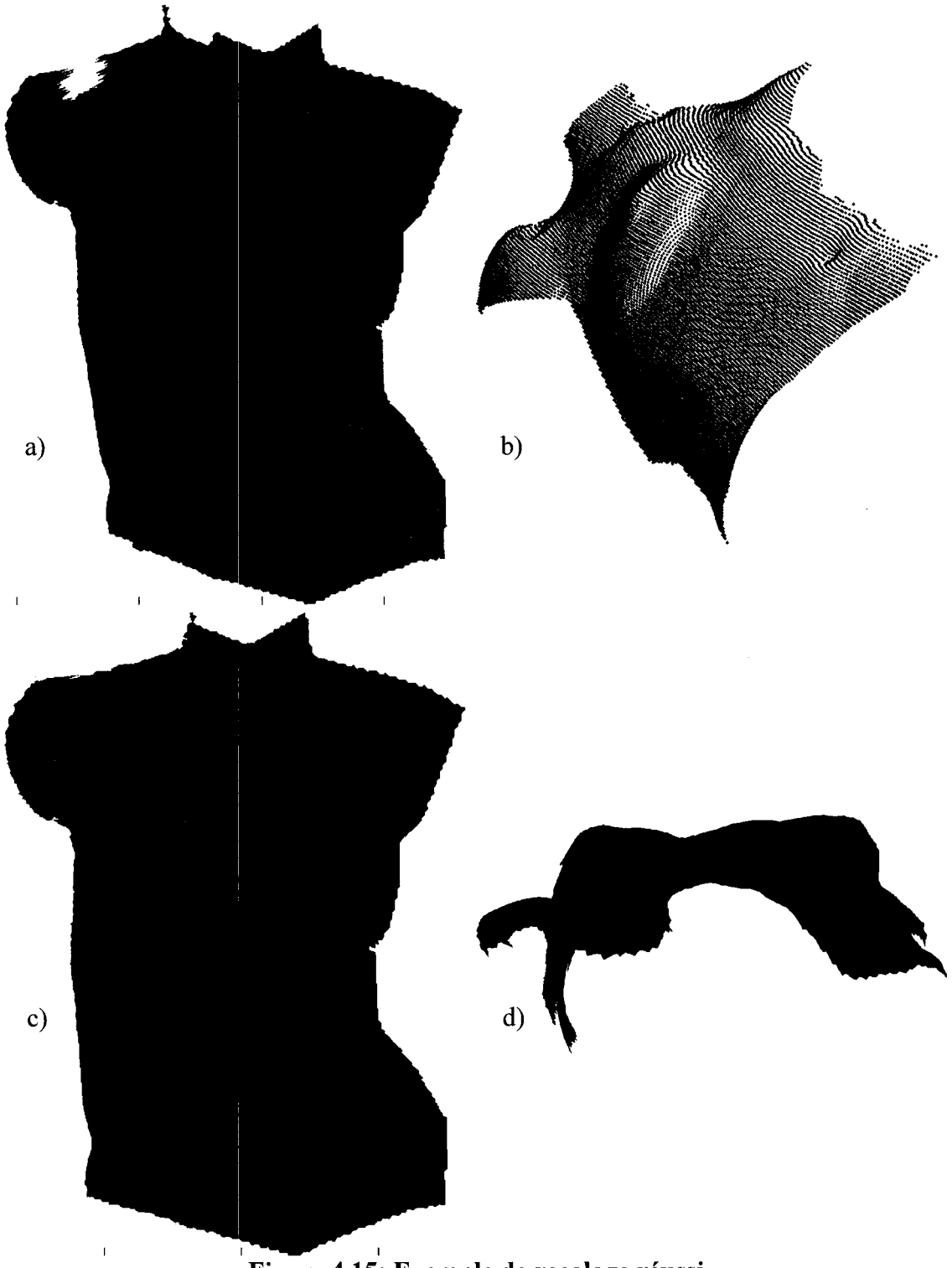


Figure 4.15: Exemple de recalage réussi

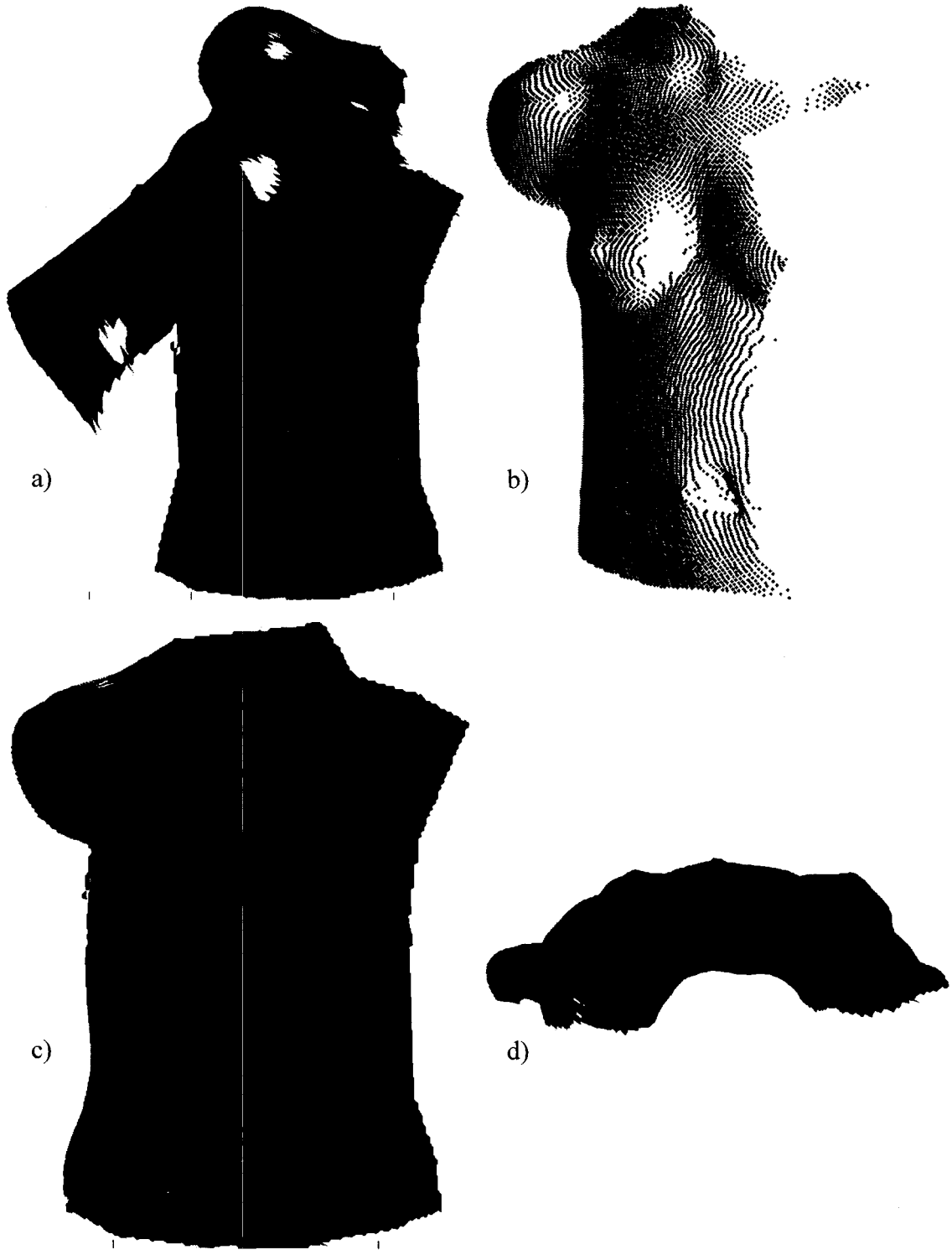


Figure 4.16: Exemple de recalage raté

4.3.3 Discussion sur le recalage des surfaces

Précédemment, nous avons vu que l'élimination des correspondances potentiellement erronées est une étape cruciale de l'algorithme IPM. Le choix du critère d'élimination devient important lorsque que tous les points d'une surface ne peuvent être appariés aux points de l'autre surface. Un raffinement possible du critère d'élimination serait de tenir compte des normales des points afin de faire un choix plus judicieux d'appariement entre les points des deux surfaces.

Aussi, pour IPM, plus la zone commune aux deux surfaces est faible, plus l'approximation initiale doit être relativement proche de la solution. Si, par exemple, une mauvaise approximation de départ comprenant une rotation permettant que le côté du torse du sujet sur une surface se trouve près du devant du torse du sujet sur l'autre surface, cela entraîne la convergence de l'algorithme IPM dans un minimum local situé très loin de la solution voulue. En effet, puisque l'algorithme ne conserve qu'une fraction des points (critère d'élimination de correspondance), il est préférable que l'approximation de départ permette l'élimination des points n'ayant pas d'équivalence sur l'autre surface, sinon, une fraction erronée sera utilisée tout au long des calculs.

Les quaternions se sont avérés relativement robustes et rapides dans toutes les situations, mais certaines rotations sont parfois difficiles à détecter. Par exemple, nos résultats ont montré qu'une rotation autour de la normale au plan approximatif d'une surface presque plane est très difficile à retrouver. En effet, deux surfaces dont une partie des points est recalée avec une distance moyenne très petite entre les points tombe rapidement dans un minimum local, comme ce serait le cas pour deux plans superposés dont un aurait subi une rotation autour de sa normale comme c'est le cas à la figure 4.17.

Enfin, la méthode IPM est relativement robuste au bruit venant déformer la surface. Malgré la forte déformation introduite à l'une et l'autre des surfaces, l'algorithme retrouve quand même une rotation qui permet de recaler les surfaces.

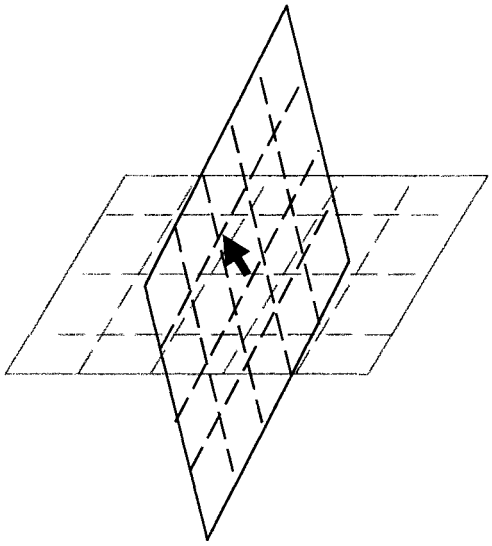


Figure 4.17: Plans avec rotation autour de la normale

Maintenant, pour ce qui est de la méthode PF, l'idée d'utiliser les déformations locales de chaque point pour effectuer un recalage est en effet très bonne, mais la technique développée dans Sun 2003 a dû être grandement améliorée avant de fournir des résultats satisfaisants. La méthode est de toute évidence un excellent moyen de détecter les points les plus intéressants de deux surfaces du point de vue du recalage, mais l'appariement des points développé ici, malgré l'utilisation de plusieurs critères successifs, demande encore à être amélioré. Les cas utilisés dans l'article sont des surfaces presque identiques et donc très faciles à recaler par IPM. À cause de cela, une bonne marge d'erreur est permise pour le recalage par la méthode PF, ce qui n'est pas le cas si les surfaces ne sont que partiellement correspondantes puisque le résultat du recalage influence le choix des points utilisés lors du raffinement par IPM.

Pour l'instant, le bruit approximativement sinusoïdal causé par le flou de mouvement diminue à la fois l'efficacité de la segmentation d'images et l'efficacité du recalage. Ce bruit fait varier les normales suffisamment pour empêcher l'algorithme PF de fonctionner à son plein potentiel. Trouver une solution pour filtrer la fréquence parasite pourrait aider à adoucir l'image suffisamment pour que le bruit sur les normales devienne relativement faible.

Filtrer les images est potentiellement plus important qu'on pourrait le croire à première vue. Si les données trop bruitées sont inutilisables pour trouver des points d'intérêt, il est très possible qu'il soit aussi difficile d'en tirer des indices cliniques. Même les données que l'on considère peu bruitées pourraient tout de même fausser le calcul d'indices cliniques.

Finalement, les images InSpeck sont bruitées près des contours des patients parce que la triangulation y est forcément moins bonne à cause de la baisse de la précision en bordure d'un objet approximativement sphérique ou cylindrique, ce qui est normal pour tout processus de triangulation. Ce bruit, un fois ajouté au flou de mouvement, rend la détection et l'appariement des points d'intérêt beaucoup plus difficile.

Chapitre 5 : Conclusion

Les méthodes développées au cours de ce projet sont certainement un atout important dans l'automatisation du processus de reconstruction 3D du tronc et pourront être utilisées pour constituer un outil clinique pour augmenter la qualité et la répétabilité des résultats. Ceci peut, sans aucun doute, permettre d'augmenter significativement l'impact clinique du suivi non-effractif de la scoliose et de faciliter les études à venir sur l'utilisation de la topographie de surface.

Le « pseudo-likelihood information criterion » (PLIC) est une technique qui, de façon générale, constitue un moyen efficace pour déterminer le nombre de segments d'une image et permet en effet de segmenter une image sans connaissances *a priori*. Le choix de la région d'intérêt à l'aide des connaissances anatomiques connues *a priori* est très simple et fournit le bon résultat lorsque la segmentation est réussie. Aussi, le nouveau système de représentation d'une image permettant de tenir compte des informations fournies par la composante de phase s'est avéré très utile et a permis d'obtenir d'excellents résultats au niveau de la précision des contours. La qualité de ces résultats aura une influence directe sur la qualité du recalage.

Le recalage est une tâche très difficile à automatiser en entier et des limites assez restreintes ont été découvertes quant à son champ d'action et ses possibilités de correction. Il est certain qu'un avenir réside en la formulation d'algorithmes « hybrides » tirant profit des avantages de plusieurs algorithmes actuellement en place, comme la combinaison de la méthode PF et de IPM. Cette combinaison est parfaitement applicable dès maintenant pour le recalage de toutes surfaces ayant une grande zone commune et un taux de bruit relativement faible sur les normales de chaque point de la surface. L'ajout d'un algorithme de segmentation automatisé comme celui présenté dans ce mémoire permettra d'assurer la maximisation de la zone commune entre les deux surfaces.

Pour l'utilisation en clinique, il serait sans doute possible, pour un montage relativement fixe, d'entrer simplement une mesure approximative de l'angle entre chaque caméra et, en calculant la transformation initiale approximative et le pourcentage de correspondance entre deux surfaces, IPM pourrait à lui seul suffire pour recaler les surfaces. Cependant, le développement d'un algorithme d'appariement de points solide permettrait non-seulement d'éviter toute intervention manuelle pour le recalage, mais permettrait en plus de commencer à songer à automatiser le recalage élastique. En effet, avec une confiance suffisamment grande dans les paires de points trouvées, il deviendrait possible de voir que deux points ayant des déformations locales semblables se retrouvent sur des surfaces légèrement différentes et il deviendrait alors possible de corriger ces surfaces pour compenser le mouvement du patient durant l'acquisition des données. La modélisation de la déformation élastique de type « respiration » pourrait d'ailleurs être effectuée et prise en considération dans les calculs.

La solution à la déformation et au flou de mouvement causé par la respiration du patient réside donc, du côté logiciel, dans la modélisation du mouvement et, à partir de cela, dans le développement d'algorithmes de recalage élastique. Du côté matériel, il faut trouver un moyen d'effectuer l'acquisition des surfaces plus rapidement.

La cause du problème de flou de mouvement est que les franges sont projetées à partir d'une petite grille qui est déplacée par un petit moteur électrique devant une source lumineuse. Non seulement le moteur peut se dérégler, se désynchroniser ou faire en sorte que les franges obtenues ne sont pas exactement séparées d'un quart de phase chacune, mais il est le facteur limitant en terme de vitesse d'acquisition.

L'équipement utilisé pour acquérir les images possède donc quelques lacunes. En plus du problème de flou de mouvement relié à la vitesse d'acquisition, plus de 25% des images sélectionnées pour effectuer la détection automatique de la région d'intérêt ont été rejetées parce que les franges sont visibles sur l'image de texture.

Les techniques de détection de région d'intérêt et de recalage utilisées au cours de ce mémoire sont suffisamment bonnes, robustes et générales pour être utilisées dans d'autres contextes. De plus l'avancement des recherches dans ces domaines permettra sûrement l'élaboration de solutions de plus en plus robustes. Cependant, dans le présent contexte, plusieurs études utilisent les mêmes données que celles utilisées dans ce mémoire et il serait probablement utile de développer un outil, comme par exemple un filtre, qui permettrait de s'assurer que les données utilisées par tous les chercheurs sont valides, tant et aussi longtemps qu'un meilleur système d'acquisition ne sera pas à notre disposition.

5.1 Recommandations

En clinique, il est fortement recommandé d'utiliser un écran, possiblement bleu ou vert, à l'arrière-plan du champ de vision de chaque caméra pour faciliter la segmentation à partir du moment où il sera envisagé d'automatiser cette étape. Le bleu et le vert se retrouvent rarement sur la peau et ce type d'écran est déjà utilisé au cinéma et à la télévision pour la création d'effets spéciaux. Durant les essais préliminaires, les objets en arrière plan ont causé beaucoup de problèmes pour le calcul du PLIC qui a dû converger trop tôt ou trop tard.

Lors des tests, les images sélectionnées répondaient autant que possibles à certains critères. Pour l'utilisation de la méthode dans plusieurs cliniques, un nouveau protocole d'acquisition respectant un ensemble de critères est proposé :

- Standardisation du positionnement des patients (bras droits, écartés à environ 45 degrés du corps est la position préférée pour l'instant)
- Pas de cheveux dans le cou
- Pas de soutien-gorge
- La présence de pantalons ou d'une jupe favorise la détection du bas du dos
- Arrière-plan monochromatique (écran bleu ou écran vert, par exemple)
- Sources de lumières provenant exclusivement des caméras autant que possible

Pour diminuer le flou de mouvement, l'utilisation d'un écran LCD transparent pour remplacer la grille pourrait sans aucun doute décupler la vitesse d'acquisition et faciliter la synchronisation entre les divers composantes électroniques. À peu près n'importe quel écran LCD pourrait être utilisé puisque ceux-ci possèdent habituellement une surface réflective en arrière plan qu'il est possible d'enlever. Pour ce qui est de la vitesse d'acquisition, le site suivant propose des caméras CCD permettant d'acquérir jusqu'à 500 millions d'images/seconde en résolution 1280 x 1240 pixels :

http://www.photonlines.com/Francais/Cameras/FCam_haute_cadence.html

Bien entendu, cela dépasse largement les besoins et les coûts d'une caméra CCD utilisée pour acquérir des images de topographies de surface, mais cet exemple sert simplement à montrer que la technologie nécessaire à l'augmentation de la vitesse d'acquisition est existante sur le marché.

Il est cependant à noter que les écrans LCD représentent une image sous forme de pixels. Cette discrétisation de l'espace pourrait empêcher de donner un déphasage de exactement 90 degrés entre les différentes franges. Le niveau de précision nécessaire au bon fonctionnement de l'appareil est à déterminer.

À plus court terme, il est recommandé de développer un filtre efficace pour les données avant d'utiliser les surfaces InSpeck dans d'autres études. Plusieurs techniques fonctionnelles pourraient être invalidées par l'utilisation de ces surfaces bruitées et il est donc primordial, si l'on désire continuer d'effectuer des études à partir de ces données, de faire en sorte que la quantité de bruit diminue à un niveau beaucoup plus bas que le niveau de bruit actuel.

Références

A.D.A.M. (Animated Dissection of Anatomy for Medicine). 2002. Scoliosis. In *Health and Age* [en ligne] http://www.healthandage.com/html/well_connected/pdf/doc68.pdf (Page consultée le 20 mai 2003)

A.D.A.M. (Animated Dissection of Anatomy for Medicine). 2003. Scoliosis. In *Yahoo! Health* [en ligne] http://health.yahoo.com/health/encyclopedia/001241/i1114.html (Page consultée le 20 mai 2003)

Bekele Araya-Yohannes. 2002. *L'utilisation des lasers de faible puissance en thérapie réflexe et en thérapie locale.* Rapport de recherche bibliographique 2001- 2002 ENSSIB (École Nationale Supérieure des Sciences de l'Information et des Bibliothèques)

Besag, J. 1986. *On the Statistical Analysis of Dirty Pictures*, Journal of the Royal Statistical Society B, Vol. 48 et No. 3, pp.259-302.

Besl P., 1989 *Chapter 1: Active optical range imaging sensors. Advances in Machine Vision* pp. 1-63, Springer-Verlag..

Bigler E. 2002. *La règle de Scheimpflug.* ENSMM (École d'ingénieurs de mécanique et des microtechniques), Besançon. In *Site de galerie-photo.* [en ligne] http://www.galerie-photo.com/demonstration-scheimpflug.html (Page consultée le 21 mai 2003).

Blais F., Beraldin A. 2001. *Tutorial: Active 3D Sensing.* Third International Conference on 3-D Digital Imaging and Modeling, Hôtel Loews Le Concorde, Québec, Canada.

Bonet J., Peraire J. 1991. *An Alternating Digital Tree (ADT) algorithm for 3D Geometric Searching and Intersection Problems*, Int. J. for Numerical Methods in Engineering, vol. 31, pp. 1-17.

Bookstein F., 1989. *Principal warps: Thin-plate splines and the decomposition of deformations*, IEEE Trans. Pattern Anal. Machine. Intell., vol.11, pp. 567-585.

Bruandet, Jean-Michel. 1996. La scoliose. In *Site de la faculté de médecine de l'université de Rennes* [en ligne]
http://noemed.univ-rennes1.fr/sisrai/art/scoliose_p_291-294.html
(Page consultée le 20 mai 2003)

Bureau de la protection contre les rayonnements des produits cliniques et de consommation. 2002a. Radiographies et grossesse. In *Site de Santé Canada*. [en ligne].
http://www.hc-sc.gc.ca/francais/vsv/aspect_medical/radiographie.html
(Page consultée le 20 mai 2003)

Bureau de la protection contre les rayonnements des produits cliniques et de consommation. 2002b. Évaluation et gestion des risques de cancer associés aux rayonnements ionisants et aux agents chimiques. In *Site de Santé Canada*. [en ligne].
http://www.hc-sc.gc.ca/hecs-sesc/pcrpcc/publication/98dhm216/chapter4.htm
(Page consultée le 20 mai 2003)

Carson C., Belongie S., Greenspan H., Malik J. 1999. *Blobworld: Image segmentation using Expectation-Maximization and its application to image querying*. Third International Conference on Visual Information Systems.

Evans A. C., Dai W., Collins L., Neelin P., et Marrett S., 1991. *Warping of a computerized 3-D atlas to match brain image volumes for quantitative neuroanatomical and functional analysis*. in Proc. SPIE 1445 Medical Imaging V: Image Processing, M. H. Loew, Ed., San Jose, CA, pp.236–246.

Family Practice Notebook. 2000. Idiopathic Scoliosis, Adolescent Scoliosis. In *Site de Family Practice Notebook*. [en ligne]. http://www.fpnotebook.com/ORT348.htm
(Page consultée le 20 mai 2003)

Fischler M.A.., Bolles R.C, 1981. *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*. Communications of the ACM, Vol. 24, No 6, pp. 381-395.

Goldberg CJ, Kaliszer M, Moore DP, Fogarty EE, Dowling FE. 2001. *Surface topography, Cobb angles, and cosmetic change in scoliosis*. Spine.

Goldberg MS, Mayo NE, Poitras B, Scott S, Hanley J. 1994. *The Ste-Justine adolescent idiopathic scoliosis cohort study. Part II : perception of health, self and body image, and participation in physical activities*. Spine. vol.19, pp.1562-1572.

Granger S., Pennec X. 2002. *Multi-scale EM-ICP: A Fast and Robust Approach for Surface Registration*, INRIA.

Hladůvka J., Gröller E. 2001. *Exploiting Eigenvalues of the {H}essian Matrix for Volume Decimation*. 9th International Conference in Central Europe on Computer Graphics, Visualization, and Computer Vision pp 124-129.

Jähne B., Haußecker H. 2000. *Computer Vision and Applications: A Guide for Students and Practitionners.* Academic Press.

Jaremko J. 2001. *Estimation of Scoliosis Severity from the Torso Surface by Neural Networks.* Thèse de doctorat, Department of Medical Science, University of Calgary.

Joncas J, Labelle H, Poitras B, Duhaime M, Rivard CH, Le Blanc R. 1996. *Douleur dorso-lombaire et scliotique idiopathique de l'adolescence.* Ann Chir. vol.50 pp.637-640.

Karachalios T, Sofianos J, Roidis N, Sapkas G, Korres D, Nikolopoulos K. 1999. *Ten-year follow-up evaluation of a school screening program for scoliosis. Is the forward-bending test an accurate diagnostic criterion for the screening of scoliosis?* Spine.

Kimmel R., Sethian J.A., 1998. *Computing Geodesic Paths on Manifolds.* Pro. Nat. Acad. Sci. pp. 8431-8435.

Kirkpatrick S., Gerlatt C. D. Jr., Vecchi M.P., 1983. *Optimization by Simulated Annealing*, Science 220, 671-680

Liu XC, Thometz JG, Lyon RM, Klein J. 2001. *Functional classification of patients with idiopathic scoliosis assessed by the Quantec system: a discriminant functional analysis to determine patient curve magnitude.* Spine.

Lorusso A., Eggert D.W., Fisher R.B. 1995. *A Comparison of Four Algorithms for Estimating 3-D Rigid Transformations*, Proceedings of the 6th British Machine Vision Conference. pp. 237-246.

Novosad, J. 2002. T*echniques explicites pour la reconstruction 3D de la colonne vertébrale à partir d'images radiographiques.* Mémoire de maîtrise, département de genie mécanique, École Polytechnique de Montréal.

Payne III WK, Ogilvie JW, Resnick MD, Kane RL, Transfeld EE, Blum RW. 1997. *Does scoliosis have a psychological impact and does gender make a difference ?.* Spine. vol. 22, pp.1380-1384.

Pazos, V. 2002. *Développement d'un système de reconstruction 3D et d'analyse de la surface externe du tronc humain pour un suivi non invasif des déformations scoliotiques.* Mémoire de maîtrise, département de genie mécanique, École Polytechnique de Montréal.

Potts, R. B. 1952. *Some generalized order-disorder transformations*, Proc. Camb. Phil. Soc., 48:106.

Qian W., Titterington D.M. 1993. *Bayesian image restoration: an application to edge-preserving surface recovery.* IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. No. 7, pp. 748 – 752.

REAMY, Brian V., SLAKEY, Joseph B. 2001. Adolescent Idiopathic Scoliosis: Review and Current Concepts. In *American Family Physician.* [en ligne]. http://www.srs.org/htm/library/review/review05.htm (Page consultée le 20 mai 2003)

Rohr K., 2001. *Landmark-Based Elastic Registration Using Approximating Thin-Plate Splines*, IEEE Transactions on Medical Imaging, Vol. 20, No. 6.

Scoliosis Research Society. 2003. In Depth Review of Scoliosis: Idiopathic Scoliosis. In *Site de Scoliosis Research Society*. [en ligne].
http://www.aafp.org/afp/20010701/111.html (Page consultée le 20 mai 2003)

Stanford D.C., Raftery A.E. 2002. *Approximate bayes factors for image segmentation: the pseudolikelihood information criterion (PLIC).* IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 24, No. 11, pp. 1517 – 1520.

Sun Y., Paik J., Koschan A., Page D.L., Abidi M.A. 2003. *Point Fingerprint: A New 3-D Object Representation Scheme*, IEEE Transactions On Systems, Man, And Cybernetics—Part B: Cybernetics, Vol. 33, No. 4.

Suzuki Nobumasa, Ono Toshiaki, Tezuka Masaki, Kamiishi Satoshi. 1992. *Moiré topography and back shape analysis – clinical application.* Int. Symposium on 3-D scoliotic deformities: joined with the VIIth International Symposium on spinal deformity and surface topography, Montréal: Editions de l'Ecole Polytechnique de Montréal.

Trucco E., Verri A. 1998. *Introductory Techniques for 3D Computer Vision.* Prentice-Hall, 480 pages.

Turner-Smith A.R. 1988. *A Television/Computer three-dimensional surface shape measurement system.* J. Biomechanics Vol 21. No 6. pp 515-529.

Turner-Smith A.R., Harris, J.D., Houghton G.R., Jefferson R.J. 1988. *A method for analysis of back shape in scoliosis.* J. Biomechanics Vol 21. No 6. pp 497-509.

University of California, San Francisco. 2003. Pediatric Orthopaedic Service. In *Site de University of California, San Francisco.* [en ligne].
http://www.ucsf.edu/orthopaedics/patientedu/peds_scoli.html
(Page consultée le 20 mai 2003)

Wahba G., 1990. *Spline Models for Observational Data.* Philadelphia, PA: Soc. Ind. Appl. Math.

Weinstein SL, Zavala DC, Ponseti IV. 1981. *Idiopathic scoliosis: long-term follow-up and prognosis in untreated patients.* J Bone Joint Surg Am. Vol. 63, pp702-712.

Weinstein Sl. 1994. *The pediatric spine : principle and practice.* Raven Press, New York.

Zhang Z., 1994. *Iterative Point Matching for Registration of Free-Form Curves,* International Journal of Computer Vision.

Zitova B., Flusser J., Kautsky J., and Peters G. 2000. *Feature Point Detection in Multiframe Images.* Proceedings of the Czech Pattern Recognition Workshop 2000 (T.Svoboda ed.), pp. 117-122, Perslak, Czech Republic. February 2000.

Annexe 1 : Explications mathématiques

L'annexe 1 comprend quelques explications mathématiques de base sur certains concepts vus lors du mémoire.

Qu'est-ce qu'un quaternion ?

Pour comprendre l'utilisation des quaternions, il faut d'abord comprendre la théorie des nombres hypercomplexes. À titre d'exemple, on fournit ici la table de multiplication (colonne * ligne) pour les nombres hypercomplexes i, j et k. Sachant cela, un quaternion peut être écrit sous la forme $q = q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} + q_4$. Aussi, souvent, on préfère le présenter sous la forme d'un vecteur et d'un scalaire ainsi : $q = (\boldsymbol{q}, q_4) = (\boldsymbol{v},s)$. On peut aussi démontrer que :

	1	**i**	**j**	**k**
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

- le conjugué d'un quaternion q est obtenu par :

$$\bar{q} = -q_1\mathbf{i} - q_2\mathbf{j} - q_3\mathbf{k} + q_4 = (-\boldsymbol{q}, q_4)$$

- la multiplication des quaternions q_a et q_b est donnée par :

$$q_a q_b = (\boldsymbol{v}_a,s_a).(\boldsymbol{v}_b,s_b) = (s_a\boldsymbol{v}_b + s_b\boldsymbol{v}_a + \boldsymbol{v}_a \boldsymbol{v}_b, s_a s_b - \boldsymbol{v}_a \boldsymbol{v}_b)$$

- l'inversion d'un quaternion :

$$q^{-1} = \frac{\bar{q}}{q\bar{q}}$$

- une rotation d'un angle θ autour du vecteur unitaire $\boldsymbol{n}$ peut s'exprimer par un seul quaternion de la façon suivante :

$$q = (\boldsymbol{v},s) = (\boldsymbol{n}\sin(\tfrac{1}{2}\theta),\cos(\tfrac{1}{2}\theta))$$

- les composantes de ce quaternion sont appelées les paramètres d'Euler. La rotation d'un quaternion $p = (\boldsymbol{p},0)$ représentant un point peut alors être obtenu par $p' = qpq^{-1}$.

Multiplicateurs de Lagrange

Les multiplicateurs de Lagrange servent à trouver l'extremum de la fonction $f(x_1,x_2,\ldots,x_N)$ lorsque soumis à une contrainte $g(x_1,x_2,\ldots,x_N) = C$ où f et g sont des fonctions possédant des dérivées partielles continues lorsque $g(x_1,x_2,\ldots,x_N) = 0$ et $\nabla g \neq 0$. Pour qu'un extremum existe, il faut vérifier que :

$$df = \frac{\partial f}{\partial x_1} dx_1 + \frac{\partial f}{\partial x_2} dx_2 + \ldots + \frac{\partial f}{\partial x_N} dx_N = 0$$

$$dg = \frac{\partial g}{\partial x_1} dx_1 + \frac{\partial g}{\partial x_2} dx_2 + \ldots + \frac{\partial g}{\partial x_N} dx_N = 0$$

En multipliant dg par λ et en additionnant df, on obtient :

$$\left(\frac{\partial f}{\partial x_1} + \lambda \frac{\partial g}{\partial x_1}\right) dx_1 + \left(\frac{\partial f}{\partial x_2} + \lambda \frac{\partial g}{\partial x_2}\right) dx_2 + \ldots + \left(\frac{\partial f}{\partial x_N} + \lambda \frac{\partial g}{\partial x_N}\right) dx_N = 0$$

Les dérivées partielles étant toutes indépendantes, on peut écrire :

$$\left(\frac{\partial f}{\partial x_k} + \lambda \frac{\partial g}{\partial x_k}\right) dx_k = 0$$

Et pour tout $k = 1,\ldots,N$, la constante λ est appelée le multiplicateur de Lagrange. Lorsque sont présentes pour f plusieurs contraintes $g_1 = 0$, $g_2 = 0$, ...

$$\nabla f = \lambda_1 \nabla g_1 + \lambda_2 \nabla g_2 + \ldots$$

Annexe 2 : Recalage élastique par splines plaque mince

On attribue généralement l'utilisation des techniques d'interpolation de type spline plaque mince à Bookstein (voir Bookstein 1989) et une des premières applications à des images tridimensionnelles à Evans et coll (1991). Plus récemment, Rohr (2001) s'est penché davantage sur l'utilisation des splines plaque mince dans le but d'effectuer du recalage élastique tout en tenant compte de l'erreur de localisation des points d'intérêts.

Interpolation de type spline plaque mince

Le problème d'interpolation de type spline plaque mince peut être formulé comme un problème d'interpolation multivarié. Soit un nombre N de points correspondants p_i et q_i, $i = 1,\ldots,N$ sur chacune des deux images de dimension d, il faut trouver une transformation continue u : $\mathbb{R}^d \rightarrow \mathbb{R}^d$ dans un espace de Hilbert , approprié de fonctions admissibles qui permet de :

1) Minimiser une distribution donnée $J = , \rightarrow \mathbb{R}$
2) Remplir les conditions d'interpolation : $q_i = u(p_i)$, $i = 1,\ldots,N$

Avec cette approche, la distribution représente l'énergie de flexion d'une plaque mince séparément pour chaque composante u_k, $k = 1,\ldots,d$ de la transformation u. Aussi, la distribution $J(u)$ peut alors être séparée en une somme de distributions, chacune d'elles ne dépendant que d'une seule composante u_k. Trouver u se décompose donc en d problèmes. Dans le cas d'une image en d dimensions et pour des dérivées d'ordre m dans la distribution, cela donne :

$$J_m^d(u) = \sum_{k=1}^{d} J_m^d(u_k)$$

Par exemple, pour une image 3D et des dérivées d'ordre 2, on obtient :

$$J_2^3(u) = \sum_{k=1}^{3} J_2^3(u_k)$$

Soit un ensemble de fonctions ϕ_v de l'espace $\Pi^{m-1}(\mathbb{R}^d)$ de tous les polynômes sur $\mathbb{R}^d$ jusqu'à l'ordre m-1, alors la dimension de cet espace est $M = (d + m - 1)!/(d!(m - 1)!)$ et $M < N$. Ceci détermine le nombre de paires de points correspondants minimum. Donc, pour une image 3D et des dérivées d'ordre 2, on obtient $M = (3 + 2 - 1)!/(3!(2 - 1)!) = 4$, alors au moins 5 paires de points correspondants sont requises pour effectuer le recalage.

La solution de minimisation de la distribution J peut alors s'écrire sous forme analytique ainsi :

$$u(x) = \sum_{\upsilon=1}^{M} a_\upsilon \phi_\upsilon(x) + \sum_{i=1}^{N} \omega_i U(x, p_i)$$

avec les fonctions de base $U(x,p_i)$ dépendantes de

1) la dimension d du domaine
2) l'ordre m des dérivées de la distribution
3) l'espace de Hilbert , des fonctions admissibles

Ceci nous donne alors :

$$U(x,p) = \begin{cases} \theta_{m,d} |x-p|^{2m-d} \ln|x-p| & si\ 2m-d\ est\ un\ entier\ positif\ pair \\ \theta_{m,d} |x-p|^{2m-d} & dans\ les\ autres\ cas \end{cases}$$

où $\theta_{m,d}$ est défini dans Wahba 1990. Par exemple, pour une image 3D et des dérivées d'ordre 2, on obtient $2m$-d = 2*2-3 = 1, donc $U(x,p) = \theta_{2,3}|x-p|$ et le noyau est donné par $\phi_1(\boldsymbol{x}) = 1$, $\phi_2(\boldsymbol{x}) = x$, $\phi_3(\boldsymbol{x}) = y$, $\phi_4(\boldsymbol{x}) = z$.

Pour déterminer les coefficients $\mathbf{a} = (a_1,\ldots,a_M)^T$ et $\mathbf{w} = (\omega_1,\ldots,\omega_N)^T$ de la solution analytique de $u(x)$, il faut résoudre le système d'équations linéaires suivant :

$$\begin{aligned} \mathbf{Kw} + \mathbf{Pa} &= \mathbf{v} \\ \mathbf{P}^T\mathbf{w} &= 0 \end{aligned}$$

où $K_{ij} = U(p_i,p_j)$, $P_{ij} = \phi_j(p_i)$ et $\mathbf{v}$ est le vecteur colonne d'une des composantes des coordonnées des points d'intérêt q_i de la seconde image. La condition $\mathbf{P}^T\mathbf{w} = 0$ représente les conditions limite et assurent que la partie élastique de la transformation tend vers 0 à l'infini.

De l'interpolation à l'approximation

Jusqu'à maintenant, on suppose que les points des deux images correspondent exactement. Cependant, puisque la plupart du temps un opérateur choisit ces points, des erreurs de localisation peuvent se produire. Pour tenir compte de ces erreurs, il faut ajouter à notre distribution un terme d'approximation quadratique, ce qui nous donne :

$$J_\lambda(u) = \frac{1}{n}\sum_{i=1}^{n}\frac{|q_i - u(p_i)|}{\sigma_i^2} + \lambda J_m^d(u_k)$$

Le premier terme de cette équation mesure la somme des distances euclidienne quadratiques entre les points appariés sur les deux images pondérée par la somme des variances combinées des points correspondants selon :

$$\sigma_i^2 = \sigma_{i,p}^2 + \sigma_{i,q}^2$$

Le second terme de la distribution mesure à quel point la transformation résultante sera lisse. Le paramètre λ sert donc de régularisation afin de déterminer le poids relatif entre une transformation précise et une transformation lisse ($\lambda > 0$). Si λ est petit, la transformation s'adapte bien aux déformations locales, alors que si λ est grand, la transformation est très lisse, mais s'adapte peu aux déformations.

Le système d'équations à résoudre est très semblable au système précédent :

$$\begin{aligned} (\mathbf{K}\mathbf{w} + n\lambda W^{-1}) + \mathbf{P}\mathbf{a} &= \mathbf{v} \\ \mathbf{P}^T\mathbf{w} &= 0 \end{aligned}$$

où

$$W^{-1} = \begin{pmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_n^2 \end{pmatrix}$$

Erreurs de localisation anisotropes

Pour l'instant des scalaires ont été utilisés pour représenter les poids qui représentent les erreurs de localisation. Cela implique que les erreurs soient strictement isotropes. Généralement, les erreurs sont différentes dans différentes directions et sont donc anisotropes.

Afin de résoudre ce problème, il suffit de remplacer les poids scalaires σ_i^2 par des poids matriciels Σ_i de dimensions $d \times d$. Pour les images 3D, il suffit de prendre les deux matrices de covariances pour produire $\Sigma_i = \boldsymbol{A}\ \Sigma_{i,p}\ \boldsymbol{A}^T + \Sigma_{i,q}$ où $\boldsymbol{A}$ est une matrice de transformation linéaire qui peut être calculée à partir des paires de points par la méthode des moindres carrés, par exemple. Si les deux images ont approximativement la même orientation et la même échelle, il suffit alors d'additionner les deux matrices de covariance. La distribution devient donc :

$$J_\lambda(u) = \frac{1}{n}\sum_{i=1}^{n}(q_i - u(p_i))^T \Sigma_i^{-1}(q_i - u(p_i)) + \lambda J_m^d(u_k)$$

Le système d'équations à résoudre demeure identique, sauf que la matrice de poids prend maintenant la forme :

$$W^{-1} = \begin{pmatrix} \Sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \Sigma_n^2 \end{pmatrix}$$

Vu ce changement, les matrices et les vecteurs du système d'équations doivent être modifiés par une matrice ou un vecteur identité ainsi :

$$K = (K_{ij} I_{3\times 3}) \quad P = (P_{ij} I_{3\times 3}) \quad w = (w_{ij} I_{3\times 1}) \quad a = (a_{ij} I_{3\times 1}) \quad v = (v_{ij} I_{3\times 3})$$

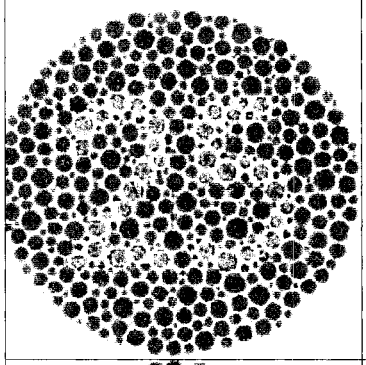
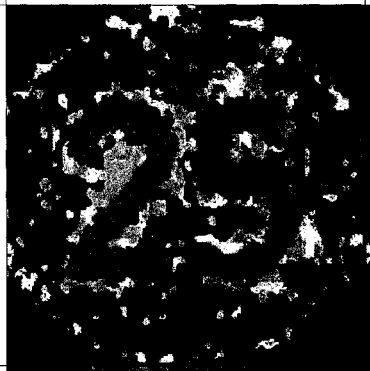
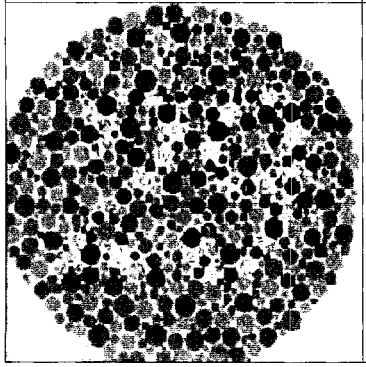
Ainsi, à l'aide de w et a, il devient possible de calculer la transformation $u(x)$. Puisque, dans ce cas-ci, le résultat sera matriciel plutôt que vectoriel, il suffit de calculer le vecteur moyenne de la matrice obtenue pour obtenir le point voulu.

Annexe 3 : Résultats de segmentation sur de images RGB

Cet annexe permet de présenter d'autres résultats de segmentation effectuées sur des images représentées selon le standard RGB afin de montrer l'efficacité de la segmentation d'images sans l'utilisation de la composante de phase.

Essais sur des images du test d'Ishihara

Le test d'Ishihara sert à dépister le daltonisme. Le test consiste à présenter des images texturées sur lesquelles des chiffres sont dessinés en utilisant différentes couleurs. Un sujet daltonien ne percevra pas les mêmes chiffres qu'un sujet sain (s'il en perçoit). Le tableau qui suit montre les résultats du PLIC/ICM avec quelques images du test d'Ishihara. La colonne « PLIC max » indique le nombre de segments optimal trouvé par l'algorithme.

Image initiale	PLIC max	Segmentation finale après les itérations de ICM	Segmentation après les opérations open et close
	7		
	6		

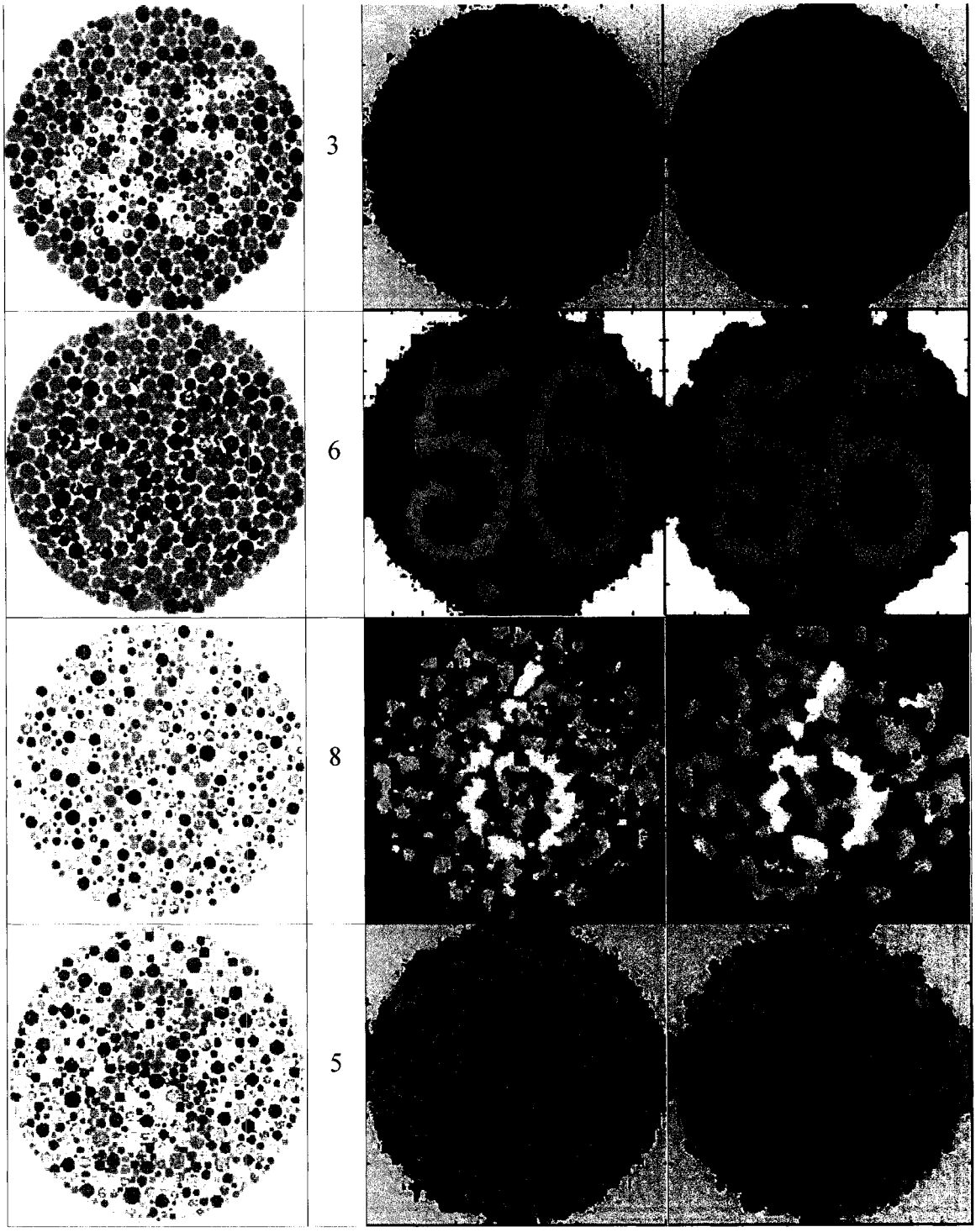


Figure A3.1 : Essais sur des images du test d'Ishihara

Il est à noter que pour initialiser les classes, la même méthode que celle décrite pour l'initialisation des classes pour les images de patients scoliotiques a été utilisée. Puisque, pour les images du test de Ishihara, les régions importantes ne sont pas les mêmes, l'initialisation revient donc plus ou moins à du hasard et ne favorise pas la segmentation sur ce type d'image en particulier.

Les couleurs des différents segments sont simplement attribuées selon la quantité de pixels de chaque classe. Les couleurs plus « froides » (bleu) sont attribuées aux segments comportant le plus grand nombre de pixels et sont ainsi attribuées par ordre décroissant de nombre de pixels jusqu'aux couleurs plus « chaudes » (rouge) qui sont attribuées aux segments comportant le moins de pixels.

Les essais sur les chiffres 25, 45 et 56 ressortent particulièrement bien. Sur l'image comportant le chiffre 29, le chiffre vu par la personne daltonienne devrait être 20 (ou 70), mais l'algorithme semble avoir détecté les deux chiffres comme faisant partie de la même classe ce qui explique que le 2^{e} chiffre soit un mélange entre un « 0 » et un « 9 ». On réussit bien à lire le 8, mais on voit bien qu'il y beaucoup d'imperfections dans l'image, possiblement causées par un manque de classes pour représenter l'image segmentée ou tout simplement par la nature de l'image. Finalement, le chiffre 6 semble être celui qui ressort le moins bien de ces tests et les opérations morphologiques « open/close » provoquent ici une « érosion » sur le chiffre 6. Il faut comprendre que ces opérations favorisent les couleurs les plus « froides » (donc les segments les plus volumineux) et que, lorsque, suite à l'attribution des couleurs des segments, le chiffre dessiné sur l'image du test d'Ishihara est représenté par des couleurs plus « chaudes » que ses voisins, ces opérations morphologiques ont tendance à le faire disparaître au profit de son voisinage. Ceci explique pourquoi on observe un « 6 » un peu flou après segmentation et que celui-ci est réduit après open/close.

Annexe 4 : Code Source

Cette section contient le code source des principales fonctions utilisées au cours de la réalisation du mémoire.

Code source de la segmentation d'images

Fichier sqdiffs.m : Calcul de la composante d'homogénéité de phase

```
function sqdiffmat = sqdiffs(W)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fonction sqdiffs
% Homogénéité des distances
% Cette fonction calcule le carré de la différence de chaque pixel
% avec ses 8 voisins et en effectue la somme. Les bordures et les coins
% sont réajustés en fonction de leur nombre de voisins.
%
% Entrées :
%       W :          Image de phase de format M*N
%
% Sorties :
%    sqdiffmat : Composante d'homogénéité de phase
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

S = size(W);
sqdiffmat = zeros(S(1), S(2));
Ma = zeros(S(1)+1, S(2)+1);
Mb = Ma;
Mc = Ma;
Md = Ma;
R = Ma;
Ma(1:S(1),1:S(2)) = W;
Mb(2:S(1)+1,2:S(2)+1) = W;
Mc(1:S(1),2:S(2)+1) = W;
Md(2:S(1)+1,1:S(2)) = W;

% Décalage diagonal pente négative
R = sqdiff(Ma,Mb);
sqdiffmat(1:S(1)-1,1:S(2)-1) = sqdiffmat(1:S(1)-1,1:S(2)-1) + R(2:S(1), 2:S(2));
sqdiffmat(2:S(1),2:S(2)) = sqdiffmat(2:S(1),2:S(2)) + R(2:S(1), 2:S(2));
% Décalage vertical
R = sqdiff(Ma,Mc);
sqdiffmat(1:S(1),1:S(2)-1) = sqdiffmat(1:S(1),1:S(2)-1) + R(1:S(1), 2:S(2));
sqdiffmat(1:S(1),2:S(2)) = sqdiffmat(1:S(1),2:S(2)) + R(1:S(1), 2:S(2));
% Décalage horizontal
R = sqdiff(Ma,Md);
sqdiffmat(1:S(1)-1,1:S(2)) = sqdiffmat(1:S(1)-1,1:S(2)) + R(2:S(1), 1:S(2));
sqdiffmat(2:S(1),1:S(2)) = sqdiffmat(2:S(1),1:S(2)) + R(2:S(1), 1:S(2));
% Décalage diagonal pente positive
R = sqdiff(Mc,Md);
sqdiffmat(2:S(1),1:S(2)-1) = sqdiffmat(2:S(1),1:S(2)-1) + R(2:S(1), 2:S(2));
sqdiffmat(1:S(1)-1,2:S(2)) = sqdiffmat(1:S(1)-1,2:S(2)) + R(2:S(1), 2:S(2));

% Ajustement des bordures et coins (moins de 8 voisins)
% -- bordure  : 5 voisins
sqdiffmat(2:end-1,[1 end]) = sqdiffmat(2:end-1,[1 end])*(8/5); % Verticales
sqdiffmat([1 end],2:end-1) = sqdiffmat([1 end],2:end-1)*(8/5); % Horizontales
% -- coin : 3 voisins
sqdiffmat([1 end],[1 end]) = sqdiffmat([1 end],[1 end])*(8/3); % Coins
```

Fichier rgb2yuv.m : Convertis une image RGB en format YUV

```
function YUV = rgb2yuv(image)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fonction rgb2yuv
% Convertit une image de format RGB en image de format YUV
%
% Entrées :
%   image : Image RGB
%
% Sorties :
%   YUV :   Image YUV
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

YUV(:,:,1) = 0.257*image(:,:,1) + 0.504*image(:,:,2) + 0.098*image(:,:,3) + 16/255;
YUV(:,:,2) = 0.439*image(:,:,1) - 0.368*image(:,:,2) - 0.071*image(:,:,3) + 128/255;
YUV(:,:,3) = -0.148*image(:,:,1) - 0.291*image(:,:,2) + 0.439*image(:,:,3) + 128/255;
```

Fichier VraisImage.m : Calcule la vraisemblance de la segmentation

```
function [vraisemblance] = VraisImage(image,theta,classe)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fonction VraisImage
% Détermine la vraisemblance de l'image reçue pour une classe donnée
% selon les paramètres de toutes les classes présentes dans l'image.
% La vraisemblance est calculée selon une estimation gaussienne
% de la distribution des pixels de chaque classe dans l'image
%
% Entrées :
%       image :     Image  de départ
%   theta :     Moyennes et matrices de covariance de chaque classe
%   classe :    Numéro de la classe courante
%
% Sortie :
%   vraisemblance : Vraisemblance de l'appartenance à la classe
%                   courante pour chaque pixel de l'image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[rows,cols,plans] = size(image);
pixels = reshape(image,rows*cols,plans)' - theta(:,1,classe)*ones(1,rows*cols); % x-u
regul = 1/((2*pi)^(plans/2)*sqrt(det(theta(:,2:end,classe))));
exposant = -0.5*dot(pixels,inv(theta(:,2:end,classe))*pixels);
vraisemblance = reshape(regul*exp(exposant),rows,cols);
```

Fichier segmenter.m

```
function [Segmentation, K]=segmenter(ImageCouleur, ImagePhase, PLICdebut, PLIClimit,
ICMlimit, phi, Te)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fonction PLIC : Pseudo likelihood information criterion
% Détermine le nombre K de segments présents dans une image et segmente l'image
%
% Entrées :
%       ImageCouleur :  Nom de l'image à segmenter
%       ImagePhase :    Nom de l'image de phase correspondante à l'image couleur
%       PLICdebut :     Nombre minimal de segments dans l'image
%                   Valeur par défaut : 4
%       PLIClimit :     Nombre maximal de segments dans l'image
%                   Valeur par défaut : 12
%       ICMlimit :      Nombre maximal d'itérations de l'algorithme ICM
%                   Valeur par défaut : 10
%       phi :           Paramètre d'homogénéité spatiale :
%                        phi < 0 -> Voisinnage des pixels tend à être similaire au pixel
```

```
%                       phi > 0 -> Voisinnage des pixels tend à être différent du pixel
%                       phi > 0 -> Voisinnage des pixels tend à être indépemdant du pixel
%                   Valeur par défaut : 10
%   Te :            Période d'échantillonnage pour l'algorithme de segmentation EM
%                   Valeur par défaut : 2
%
% Sorties :
%   Segmentation :  Image segmentée en K classes
%   theta :         Moyenne et covariance de chaque classe
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Vérification des paramètres
if nargin < 2
    error('La fonction prends entre 2 et 7 arguments. Pour l''aide, tapez ''help
segmenter''')
elseif nargin >= 2 & nargin < 7
    Te = 2;
    if nargin < 6
        phi = 10; % Optimisé qualitativement expérimentalement
        if nargin < 5
            ICMlimit = 10;
            if nargin < 4
               PLIClimit = 12;
                if nargin < 3
                      PLICdebut = 4;
                end
            end
        end
    end
end

% Lecture de l'image couleur et conversion RGB -> YUV
Img = imread(ImageCouleur,'bmp');
Img = im2double(Img);

if not(isempty(ImagePhase))
       Img = rgb2yuv(Img);
       % Filtrage, puis affichage de l'image [Homogénéité, U, V]
       ImgP = ReadBMP(ImagePhase,1); % Lecture de l'image de phase
       W = sqdiffs(ImgP); % Calcul de l'homogénéité des distances

       % Normaliser : toutes les valeurs doivent être comprises entre 0 et 1
       W = (W - min(min(W))) / (max(max(W)) - min(min(W)));
    W =  medfilt2(W);
       Img(:,:,1)=W;
end

figure;
image(Img);

% Trouver le nombre de segments
[Segmentation, theta] = PLIC(Img, phi, PLICdebut, PLIClimit, ICMlimit);
% Segmenter avec le nombre de segments trouvés
S = size(theta);
[Segmentation, theta] = ICM(Img,S(3),phi,ICMlimit,Te);

% Donner les couleurs plus foncées aux classes les plus importantes
Taille = zeros(S(3),1);
for m=1:S(3)
    SSegm = size(find(Segmentation == m)); % Nombre d'occurences de chaque classe
    Taille(m) = SSegm(1);
end
```

```
tmpSegm = zeros(size(Segmentation));
for m=1:S(3)
    mcur = find(Taille == max(Taille)); % classe ayant présentement le maximum
d'occurences
    indices = find(Segmentation == mcur(1)); % pixels de la classe courante
    tmpSegm(indices) = m;
    Taille(find(Taille == max(Taille))) = -1; % retirer la classe des choix possibles
end
Segmentation = tmpSegm;

[plans,params,K] = size(theta);
```

Ficher segmentationEM.m : Segmentation Expectation-Maximization

```
function [theta]=segmentationEM(image,classes,Te)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fonction segmentationEM : Expectation Maximisation
% Effectue la segmentation d'une image.
% *** Important -> On suppose un segment important au centre de l'image. ***
%
% Entrées :
%       image :     Image à segmenter :
%         Matrice de N*M*C où C est la dimension du vecteur d'attributs au point (n,m).
%         L'image a une représentation multispectrale (RGB, par exemple).  Les différentes
%         composantes sont représentés par la troisième dimension de l'image.
%   classes :   Nombre de segments présents dans l'image (au moins 2)
%   Te :        Période d'échantillonnage de l'image (identique en x et en y)
%               Valeur par défaut : 2
%
% Sorties :
%   Segmentation :  Image segmentée en K classes
%   theta :         Moyenne et covariance de chaque classe
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 2
    error('La fonction prends 2 ou 3 arguments. Pour l''aide, tapez ''help
segmentationEM''');
elseif nargin == 2
    Te = 2;
end

if ndims(image) < 3
    error('L''image doit avoir au moins 3 dimensions. Pour l''aide, tapez ''help
segmentationEM''');
elseif ndims(image) > 3
    warning('L''image a plus de 3 dimensions. Pour l''aide, tapez ''help
segmentationEM''');
end

if classes < 2
    error('La segmentation peut s''effectuer pour un minimum de 2 classes');
end

[rows,cols,plans] = size(image);

% On prend une image sous-échantillonnée (approche multi-résolution)
% Sous-échantillonnage par filtre gaussien + 1 point sur 8 (en commentaires)
%   sigma = Te / 2;
%   TailleFilt = 2*Te+1;
%   h = fspecial('gaussian',TailleFilt,sigma);
sImg = size(image);
for m = 1:sImg(3)
    % ImgFiltree = imfilter(image,h);
```

```
    SousEch(1:sImg(1)/Te,1:sImg(2)/Te,m) = image(1:Te:end,1:Te:end,m); % Sous-
échantillonnage non-filtré
end

sSE = size(SousEch); % Dimensions du sous-échantillon
% Toute premiere segmentation effectuée selon un modele géométrique constant
Carte=ApproxGeom(sSE(1), sSE(2), classes);

SousEch=reshape(SousEch,sSE(1)*sSE(2),sSE(3));
Carte=reshape(Carte,sSE(1)*sSE(2),1);

% Initialiser les variables
theta = zeros(plans,plans+1,classes);
logVraisPrec = Inf;
NewEch = reshape(SousEch,sSE(1),sSE(2),plans);
vraisemblance(sSE(1),sSE(2),classes)=0;
CarteTmp(sSE(1)*sSE(2))=0;
CarteOld(sSE(1)*sSE(2))=0;
vraisemblancemax(sSE(1),sSE(2))=0;

% Itérer
index = 0;
for iteration = 1:100
    vraisemblance(:,:,:)=-1;
    vraisemblancemax(:,:)=-1;
    CarteTmp(:)=0;
    for k = 1:classes
        index = find(Carte == k);
        if size(index) == 0
            break;
        end
        theta(:,1,k)=mean(SousEch(index,:));
        theta(:,2:end,k)=cov(SousEch(index,:));
        % Retour a la segmentation précédente si on rencontre
        % une matrice singulière dans le cas sans aucune variance (fond uni ou 1 seul
pixel)
        if det(theta(:,2:end,k)) < eps
            break;
        end
        % On suppose les poids uniforme pour tous les attributs (aucune connaissance a
priori)
        vraisemblance(:,:,k) = VraisImage(NewEch,theta,k);
        vraisemblancemax = max(vraisemblance(:,:,k),vraisemblancemax);
        CarteTmp(find(vraisemblancemax == vraisemblance(:,:,k)))=k; % Mise à jour de la
carte
    end
    if size(index) == 0 | det(theta(:,2:end,k)) < eps
%         figure;
%         imagesc(reshape(Carte,sSE(1),sSE(2)));
        Carte = CarteOld;
        break;
    end
    CarteOld = Carte;
    Carte = CarteTmp;

    logVrais = sum(sum(log((1/plans)*sum(vraisemblance,3))));
       if abs(logVraisPrec-logVrais)/logVrais <= 0.01 % Critère d'arrêt : 1% de
différence
        break;
       end
    logVraisPrec=logVrais;
    % Affichage de l'itération courante
    message = sprintf('EM, Iteration %d', iteration);
```

```
    disp(message);
end

% Recalcul du theta sans l'information spatiale
SousEch = SousEch(:,1:sSE(3));
NewEch = reshape(SousEch,sSE(1),sSE(2),sSE(3));
theta = zeros(sSE(3),sSE(3)+1,classes);
for k = 1:classes
    index = find(Carte == k);
    theta(:,1,k)=mean(SousEch(index,:));
    theta(:,2:end,k)=cov(SousEch(index,:));
    % Ajout d'une matrice diagonale pour éviter d'obtenir
    % une matrice singulière dans le cas sans aucune variance (fond uni)
    if det(theta(:,2:end,k)) == 0
        theta(:,2:end,k) = theta(:,2:end,k) + 1e-10*eye(plans);
        disp 'Matrice de covariance singulière - ajustement effectué'
    end
end
```

Ficher segmentationMMAP.m : Segmentation Maximum Marginal A Posteriori

```
function res = SegmentationMMAP(image,theta,SegPrec,phi)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fonction SegmentationMMAP : Maximum marginal a posteriori
% Effectue la segmentation d'une image au sens du maximum de vraisemblance.
%
% Entrées:
%       image : Image à segmenter :
%       Matrice de N*M*C où C est la dimension du vecteur d'attributs au point (n,m).
%       L'image a une représentation multispectrale (RGB, par exemple). Les
%       différentes composantes sont représentés par la troisième dimension de l'image
%   theta :     Paramètres des distributions de chaque classe.
%       Cette fonction assume le modèle de distribution gaussien.
%       1re colonne : moyenne de chaque composante
%       2e colonne et + : matrice de covariance des composantes
%       Les différents plans de theta représentent les différentes composantes de l'image
%   SegPrec :   Approximation antérieure de la segmentation de l'image ( utilisé pour
%               le modele de Potts ).
%       phi :       Paramètre d'homogénéité spatiale :
%                    phi < 0 -> Voisinnage des pixels tend à être similaire au pixel
%                    phi > 0 -> Voisinnage des pixels tend à être différent du pixel
%                    phi > 0 -> Voisinnage des pixels tend à être indépemdant du pixel
%               Valeur par défaut : 10
% Sortie :
%   res :       Matrice indiquant à quelle classe appartient chaque pixel de l'image.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 3
    error('La fonction prends 3 ou 4 arguments. Pour l''aide, tapez ''help
SegmentationMMAP''');
elseif nargin == 3
    phi = 10;
end

if ndims(image) < 3
    error('L''image doit avoir au moins 3 dimensions. Pour l''aide, tapez ''help
SegmentationMMAP''');
elseif ndims(image) > 3
    warning('L''image a plus de 3 dimensions. Pour l''aide, tapez ''help
SegmentationMMAP''');
end

[rows,cols,plans] = size(image);
[plans2,params,classes] = size(theta);
```

```
if (plans ~= plans2)
    error('Le nombre de composantes de l''image et de theta ne concordent pas')
end

% Calculer la vraisemblance de chaque classe à chaque pixel
for k = 1:classes % Boucle sur chaque classe
    CarteK = SegPrec==k;
    if (length(CarteK)~=0)
        Nk(k,:,:) = [CarteK(2:rows,2:cols),zeros(rows-1,1);zeros(1,cols)] + ...
                    [zeros(rows-1,1),CarteK(2:rows,1:cols-1);zeros(1,cols)] + ...
                    [zeros(1,cols);zeros(rows-1,1),CarteK(1:rows-1,1:cols-1)] + ...
                    [zeros(1,cols);CarteK(1:rows-1,2:cols),zeros(rows-1,1)] + ...
                    [CarteK(2:rows,1:cols);zeros(1,cols)] + ...
                    [zeros(1,cols);CarteK(1:rows-1,1:cols)] + ...
                    [CarteK(1:rows,2:cols),zeros(rows,1)] + ...
                    [zeros(rows,1),CarteK(1:rows,1:cols-1)];
        L(k,1:rows,1:cols) = VraisImage(image,theta,k);
    else
        L(k,1:rows,1:cols) = 0;
    end
end

% Déterminer la classe de chaque pixel, donc la classe ayant la plus grande vraisemblance
[dummy,map] = max(L.*exp(phi*Nk));
res = reshape(map,rows,cols);
```

Ficher segmentationMV.m : Segmentation au sens du Maximum de Vraisemblance

```
function res = SegmentationMV(image,theta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fonction SegmentationMV : maximum de vraisemblance
% Effectue la segmentation d'une image au sens du maximum de vraisemblance.
%
% Entrées:
%       image :     Image à segmenter :
%                   Matrice de N*M*C où C est la dimension du vecteur d'attributs au
point (n,m).
%                   L'image a une représentation multispectrale (RGB, par exemple).  Les
différentes
%                   composantes sont représentés par la troisième dimension de l'image.
%   theta :     Paramètres des distributions de chaque classe.
%               Cette fonction assume le modèle de distribution gaussien.
%               1re colonne : moyenne de chaque composante
%               2e colonne et + : matrice de covariance des composantes
%    * Les différents plans de theta représentent les différentes composantes de l'image
%
% Sortie :
%   res :       Matrice indiquant à quelle classe appartient chaque pixel de l'image.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 2
    error('La fonction prends 2 arguments. Pour l''aide, tapez ''help SegmentationMV''');
end

if ndims(image) < 3
    error('L''image doit avoir au moins 3 dimensions. Pour l''aide, tapez ''help
SegmentationMV''');
elseif ndims(image) > 3
    warning('L''image a plus de 3 dimensions. Pour l''aide, tapez ''help
SegmentationMV''');
end

[rows,cols,plans] = size(image);
[plans2,params,classes] = size(theta);
```

```
if (plans ~= plans2)
    error('Le nombre de composantes de l''image et de theta ne concordent pas')
end

% Calculer la vraisemblance de chaque classe a chaque pixel
for k = 1:classes   % Boucle sur chaque classe
    L(k,1:rows,1:cols) = VraisImage(image,theta,k);
end

% Déterminer la classe de chaque pixel
[dummy,map] = max(L);
res = reshape(map,rows,cols);
```

Fichier PLIC.m : Algorithme PLIC

```
function [Segmentation,theta] = PLIC(image, phi, debut, limit, ICMlimit)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fonction PLIC : Pseudo likelihood information criterion
% Détermine le nombre K de segments présents dans une image et segmente l'image
%
% Entrées :
%       image :     Image à segmenter :
%                   Matrice de N*M*C où C est la dimension du vecteur d'attributs au
point (n,m).
%                   L'image a une représentation multispectrale (RGB, par exemple).  Les
différentes
%                   composantes sont représentés par la troisième dimension de l'image.
%       phi :       Paramètre d'homogénéité spatiale :
%                   phi < 0 -> Voisinnage des pixels tend à être similaire au pixel
%                   phi > 0 -> Voisinnage des pixels tend à être différent du pixel
%                   phi > 0 -> Voisinnage des pixels tend à être indépemdant du pixel
%               Valeur par défaut : 10
%   debut :     Valeur initiale pour le PLIC
%               Valeur par défaut : 4
%       limit :     Nombre maximal de segments dans l'image
%               Valeur par défaut : 12
%       ICMlimit :  Nombre maximal d'itérations de l'algorithme ICM
%               Valeur par défaut : 10
%
% Sorties :
%   Segmentation :  Image segmentée en K classes
%   theta :         Moyenne et covariance de chaque classe
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin == 1
    phi = 10;
    debut = 4;
    limit = 12;
    ICMlimit = 10;
elseif nargin < 4
    error('La fonction prends 1 ou 4 arguments. Pour l''aide, tapez ''help PLIC''');
end

if ndims(image) < 3
    error('L''image doit avoir au moins 3 dimensions. Pour l''aide, tapez ''help
PLIC''');
elseif ndims(image) > 3
    warning('L''image a plus de 3 dimensions. Pour l''aide, tapez ''help PLIC''');
end

if debut > limit
    error('Le nombre de segments choisi pour l''itération initiale dépasse la limite !')
end
```

```
if limit > 12
    limit = 12;
    warning('Limite des itérations choisie plus grande que 12. Limite remise a 12.');
end

% Sous-échantillonnage de l'image selon nombre de pixels
[rows,cols,plans] = size(image);
Te = ceil(sqrt(rows*cols / 150000)); % Règle du pouce pour la période d'échantillonnage
(déterminée expérimentalement)
image = image(1:Te:end,1:Te:end,:);
sImg = size(image);
rows = sImg(1);
cols = sImg(2);
N = rows*cols;

% Initialisation des variables
PlicPrec = -inf;
X = zeros(rows,cols);
theta = 0;

% Début de l'algorithme PLIC avec K = 2
for K = debut:limit
    % Conserver les valeurs de l'itération précédente
    Xprec = X;
    thetaPrec = theta;

    % Appelle l'algorithme récursif ICM
    [X,theta] = ICM(image,K,phi,ICMlimit,2);

    % Calcul du voisinnage 8 de chaque pixel
        for m = 1:K
        CarteK = X==m;
        Nk(m,:,:) = [CarteK(2:rows,2:cols),zeros(rows-1,1);zeros(1,cols)] + ...
                    [zeros(rows-1,1),CarteK(2:rows,1:cols-1);zeros(1,cols)] + ...
                    [zeros(1,cols);zeros(rows-1,1),CarteK(1:rows-1,1:cols-1)] + ...
                    [zeros(1,cols);CarteK(1:rows-1,2:cols),zeros(rows-1,1)] + ...
                    [CarteK(2:rows,1:cols);zeros(1,cols)] + ...
                    [zeros(1,cols);CarteK(1:rows-1,1:cols)] + ...
                    [CarteK(1:rows,2:cols),zeros(rows,1)] + ...
                    [zeros(rows,1),CarteK(1:rows,1:cols-1)];
    end

    % Calcul de la somme de toutes nos exponentielles qui permettra de calculer
    % le pourcentage du voisinnage occupé par chaque classe
        denom = reshape(sum(exp(phi*Nk),1),rows,cols);

        % Calcul de la somme
        somme_fYpX = zeros(rows,cols);
        for m = 1:K
        Potts = reshape(exp(phi*Nk(m,:,:)),rows,cols)./denom; % Voisinnage pour une
classe donnée
        somme_fYpX = somme_fYpX + VraisImage(image,theta,m).*Potts; % Pour le calcul du
PLIC
    end

    % dimension de theta
        S = size(theta);

        % calcul du PLIC
        logL = log(somme_fYpX);
    dklogN = K*prod(S)*log(N);
    LxYK = sum(sum(logL));
        PlicCur = 2*LxYK - dklogN;
```

```
    if (PlicCur < PlicPrec)
        % Affichage des résultats numériques
%         PlicCur
%         LxYK
%         dklogN
        Segmentation = Xprec; % On reprends la segmentation précédente jugée préférable
        theta = thetaPrec; % On reretourne aussi les paramètres des classes
        message = sprintf('PLIC optimal déterminé pour %d segments', K-1);
        disp(message);
        break; % Fin de l'algorithme
    end
    PlicPrec = PlicCur;

    % Affichage des résultats à chaque étape
%     figure;
%     imagesc(X,[1 K]);
%     title(sprintf('PLIC %d segments, PLIC=%g, sommelog=%g, dklogN=%
g',K,PlicCur,LxYK,dklogN));
%     drawnow;

    Segmentation = X;

    % Affichage de l'itération courante
    message = sprintf('PLIC pour %d segments calculé', K);
    disp(message);
end
```

Fichier ICM.m : Algorithme ICM

```
function [Segmentation,theta] = ICM(image,classes,phi,NbIt,Te,theta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Fonction ICM : Iterated Conditionnal Modes
% Détermine le nombre K de segments présents dans une image et segmente l'image
%
% Entrées :
%       image :     Image à segmenter :
%                   Matrice de N*M*C où C est la dimension du vecteur d'attributs au
point (n,m).
%                   L'image a une représentation multispectrale (RGB, par exemple).  Les
différentes
%                   composantes sont représentés par la troisième dimension de l'image.
%   classes :   Nombre de segments présents dans l'image (au moins 2)
%       phi :       Paramètre d'homogénéité spatiale :
%                   phi < 0 -> Voisinnage des pixels tend à être similaire au pixel
%                   phi > 0 -> Voisinnage des pixels tend à être différent du pixel
%                   phi > 0 -> Voisinnage des pixels tend à être indépemdant du pixel
%               Valeur par défaut : 10
%       NbIt :      Nombre maximal d'itérations de l'algorithme ICM
%               Valeur par défaut : 10
%   Te :        Période d'échantillonnage de l'image (identique en x et en y)
%               Valeur par défaut : 2
%   theta :     *** La segmentation EM est utilisée si le paramètre n'est pas fourni ***
%               Paramètres des distributions de chaque classe.
%               Cette fonction assume le modèle de distribution gaussien.
%               1re colonne : moyenne de chaque composante
%               2e colonne et + : matrice de covariance des composantes
%               Les différents plans de theta représentent les différentes composantes de
l'image
%
% Sorties :
%   Segmentation :  Image segmentée en K classes
%   theta :         Moyenne et covariance de chaque classe
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
if nargin < 5 & nargin ~= 2
    error('La fonction prends 2,5 ou 6 arguments. Pour l''aide, tapez ''help ICM''');
elseif nargin == 2
    phi = 10;
    NbIt = 10;
    Te = 2;
end

if ndims(image) < 3
    error('L''image doit avoir au moins 3 dimensions. Pour l''aide, tapez ''help ICM''');
elseif ndims(image) > 3
    warning('L''image a plus de 3 dimensions. Pour l''aide, tapez ''help ICM''');
end

if classes < 2
    error('La segmentation peut s''effectuer pour un minimum de 2 classes');
end

% Initialisation des variables
[rows,cols,plans] = size(image);
Pixels = reshape(image,rows*cols,plans);

% Premiere approximation par segmentation EM si nécessaire
if nargin < 6
    [theta] = segmentationEM(image, classes, Te);
end

% Initialiser les deux segmentations précédentes
Segmentation = SegmentationMV(image,theta);
SegmentationAnt = Segmentation;
SegmentationAntAnt = SegmentationAnt;

% Début de l'algorithme itératif ICM
for It = 1:NbIt
    % Approximation de theta ( hypothèses de stationnarité et d'ergodisme )
    SegmentationList = reshape(SegmentationAnt,rows*cols,1);
    for k = 1:classes
        index = find(SegmentationList == k);
        if size(index) == 0 % Aucun élément dans cette classe
            theta(:,1,k)=0;
            theta(:,2:end,k)=0;
        else
            theta(:,1,k)=mean(Pixels(index,:));
            theta(:,2:end,k)=cov(Pixels(index,:));
        end
        % Ajout d'une matrice diagonale pour éviter d'obtenir
        % une matrice singulière dans le cas sans aucune variance (fond uni)
        if det(theta(:,2:end,k)) < eps
            theta(:,2:end,k) = theta(:,2:end,k) + 1e-10*eye(plans);
            disp 'Matrice de covariance singulière - ajustement effectué'
            if size(index) < 0.01*rows*cols % La classe sans variance prends moins de 1%
de la surface totale
                return;
            end
        end
    end

    % Effectuer la segmentation du maximum marginal a posteriori
    Segmentation = SegmentationMMAP(image,theta,SegmentationAnt,phi);

    % Si l'image est à 99% identique à une des 2 itérations précédentes, on cesse ICM
    if ((sum(sum(Segmentation ~= SegmentationAnt)) < (0.01*prod(size(Segmentation))))
| ...
```

```
        (sum(sum(Segmentation ~= SegmentationAntAnt)) < (0.01*prod(size(Segmentation)))))
        break;
    end

    % Conserver les deux segmentations précédentes
    SegmentationAntAnt = SegmentationAnt;
    SegmentationAnt = Segmentation;

    % Affichage de l'itération courante
    message = sprintf('ICM, Iteration %d', It);
    disp(message);
end
```

Fichier CouperMembres.m : Détection des aisselles et raffinement de la région d'intérêt

```
function image = CouperMembres(image)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fonction CouperMembres
% Coupe le cou et les bras de la personne sur l'image
%
% Entrées :
%       image :     Image binaire contenant des 1 pour le corps et des 0 pour le reste
%
% Sortie :
%   image :     Image sans le cou et sans les bras
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Pré-traitement
im1 = image(1:end-1,:) + image(2:end,:);
im1(find(im1 == 2)) = 0;
sumI = ceil(sum(im1,1)/2);
% Adoucir la somme en prenant la moyenne des 5 voisins
sumI = [sumI 0 0 0 0] + [0 sumI 0 0 0] + [0 0 sumI 0 0] + [0 0 0 sumI 0] + [0 0 0 0
sumI];
sumI = floor(sumI(3:end-2)/5);

% Couper le cou
sumX = sum(image,1);
imaxX = find(sumX == max(sumX)); % Largeur du corps
imaxX = min(imaxX);
if sumI(imaxX) >= 2 % Si le max est sous les aisselles (au moins une)
    imaxX = max(find(sumI(1:imaxX) <= 1));
end

iminX = max(find(sumX(1:imaxX) < max(sumX)/4)); % Le cou au plus le quart de la largeur
corps
iminX = min(iminX,max(find(sumI(1:imaxX) > 1))+1); % Stabilisation de l'image
if size(iminX) == 0 % Si tout est bien lisse du cou aux épaules
    iminX = max(find(sumX(1:imaxX) == 0))+1;
    if size(iminX) == 0 % Si aucun 0
        iminX = 1;
    end
end

icutX = max(find(sumX(iminX:imaxX) < max(sumX)/6))+iminX; % cou ~ 1/6 corps
if size(icutX) == 0
    icutX = iminX;
end

image(:,1:icutX) = 0;

figure;
```

```
imagesc(image);
title('Moins le cou');
drawnow;

% Couper les bras
icutX1 = min(find(sumI(imaxX:end) >= 2))+imaxX;
icutX2 = min(find(sumI(imaxX:end) >= 3))+imaxX;
i1 = image(1:end-1,icutX1);
i2 = image(2:end,icutX1);
ygauche1 = min(find((i1-i2) == 1));
ydroit1 = max(find((i2-i1) == 1));

i1 = image(1:end-1,icutX2);
i2 = image(2:end,icutX2);
ygauche2 = min(find((i1-i2) == 1));
ydroit2 = max(find((i2-i1) == 1));

if abs(ygauche1 - ygauche2) < abs(ydroit1 - ydroit2)
    ydroit = ydroit2;
    icutXd = icutX2;
    if ((abs(ygauche1 - ygauche2))/ygauche1) < 0.05
        ygauche = ygauche1;
        icutXg = icutX1;
    else
        ygauche = ygauche2;
        icutXg = icutX2;
    end
else
    ygauche = ygauche2;
    icutXg = icutX2;
    if ((abs(ydroit1 - ydroit2))/ydroit1) < 0.05
        ydroit = ydroit1;
        icutXd = icutX1;
    else
        ydroit = ydroit2;
        icutXd = icutX2;
    end
end

% Couper en diagonale
if ygauche < icutXg
        cutg = 2*triu(ones(ygauche));
    dimy = (icutXg - ygauche + 1);
        image(1:ygauche,dimy:icutXg) = image(1:ygauche,dimy:icutXg) + cutg;
else
        cutg = 2*triu(ones(icutXg));
    dimy = (ygauche - icutXg + 1);
        image(dimy:ygauche,1:icutXg) = image(dimy:ygauche,1:icutXg) + cutg;
end

s = size(image);
dimy = (s(1) - ydroit + 1);
if dimy < icutXd
        cutd = 2*tril(ones(dimy));
    cutd = cutd(:,end:-1:1);
        image(ydroit:end,(icutXd-dimy+1):icutXd) = image(ydroit:end,(icutXd-dimy+1):
icutXd) + cutd;
else
        cutd = 2*tril(ones(icutXd));
        cutd = cutd(:,end:-1:1);
        image(ydroit:end-(dimy-icutXd),1:icutXd) = image(ydroit:end-(dimy-icutXd),
1:icutXd) + cutd;
end
```

```
image(find(image > 1)) = 0;
figure;
imagesc(image);
title('Moins les bras');
drawnow;

xfin = max(icutXd,icutXg);
yfin = ceil((ygauche+ydroit)/2);
while image(yfin,xfin) == 0
    xfin = xfin + 1;
    if yfin > s(2)
        image = zeros(s);
        return;
    end
end
[bw1,idx]=bwfill(not(image),xfin,yfin,4);
image = zeros(s);
image(idx) = 1;
```

Code source du recalage

Fichier reechantillonne.m : Rééchantillonnage de point fingerprints

```
function [pf_out, pts]=reechantillonne(pf_in, NbEch, mode)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fonction reformatPF
% Change le tableau 2D de sortie de la fonction C en tableau 3D
%
% Entrees : pf_in : Tableau de courbes en N*4. Chaque ligne contient :
%                   No du point  |  No de la courbe  |  Taille du rayon | Angle du rayon
%           NbEch : Nombre d'echantillons dans une courbe
%           mode  : Mode d'interpolation
%                   1 = Linéaire (défaut)
%                   2 = Par Splines Cubiques
%
% Sorties :
%   pf_out  : Tableaux de courbes (Point Fingerprints)
%             Dimensions : nbech * nbpts * nbcrb
%                                                              Ordre de grandeur
%             nbech = nombre d'échantillons             |        100
%             nbcrb = nombre de courbes par échantillon |        5
%             nbpts = nombre de points par courbe       |        32
%             Donc, par exemple : 100*32*5
%
%   pts     : Points correspondants dans les courbes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 3
    mode = 1;
else
    if(mode ~= 1 & mode ~= 2)
        mode = 1;
    end
end

NbRayons = max(pf_in(:,2));
Incr = 2*pi/NbEch;
XX = 0:Incr:(NbEch-1)*Incr;
pts = unique(pf_in(:,1));
NbPts = size(pts,1);
pf_out = zeros(NbPts,NbEch,NbRayons);
for m = 1:NbPts
```

```
    indices = find(pf_in(:,1) == pts(m)); % Indices pour le bon point
    for n = 1:NbRayons
        indices2 = indices(find(pf_in(indices,2) == n)); % Indices parmi les indices du
point pour le bon rayon
        indices2 = [indices2(end);indices2;indices2(1)];
        rayons = pf_in(indices2,3);
        angles = pf_in(indices2,4);
        angles(1) = angles(1) - 2*pi;
        angles(end) = angles(end) + 2*pi;
        if mode == 1
            index = 1;
            for ech = 1:NbEch
                angle = (ech-1)*Incr;
                while angles(index) < angle
                    index = index + 1;
                end
                anglea = angles(index - 1);
                angleb = angles(index);
                pf_out(m,ech,n) = ((angleb - angle)*rayons(index - 1)+(angle - anglea)
*rayons(index))/(angleb - anglea);
            end
        else
            cs = spline(angles,[0 rayons' 0]);
            pf_out(m,:,n) = ppval(cs,XX);
        end
    end
end
```

Fichier appariement.m : Appariement des point fingerprints sélectionnés

```
function [papp1,papp2,indices1,indices2]=appariement(courbes1,courbes2,pts1,pts2,dc1,dc2)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fonction appariement
% Trouve les points correspondants grâce aux Point Fingerprints
%
% Entrées :
%   courbes1, courbes1  : Tableaux de courbes (Point Fingerprints)
%                     Dimensions : nbech * nbpts * nbcrb
%                                                          Ordre de grandeur
%                     nbech = nombre d'échantillons      |       100
%                     nbcrb = nombre de courbes par échantillon  |       5
%                     nbpts = nombre de points par courbe        |       32
%
%   pts1, pts2          : Points correspondants dans les courbes
%
% Sortie :
%   papp1, papp2        : Points appariés des deux mesh
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin < 4
    error('La fonction prends entre 4 arguments. Pour l''aide, tapez ''help
appariement''')
end

% Nombre de courbes par échantillon
nc = size(courbes1,3);
if nc ~= size(courbes2,3);
    error('Nombre inégal de courbes par échantillon');
end

% Nombre de points par courbe
np = size(courbes1,2);
if np ~= size(courbes2,2);
    error('Nombre inégal de points par courbes pour les deux mesh');
```

```
end

% Établir les correspondances
results = zeros([size(courbes2,1) size(courbes1,2) size(courbes1,1)]);
for m = 1:nc
    results = results + xcorrperiod(courbes1(:,:,m),courbes2(:,:,m));
end

% Trouver les meilleures correlations
index = 1;
nb_result = 30;
while (max(results(:)) > 0)
   x = find(results(:) == max(results(:))); % No de vecteur de a et décalage + 1
   [vectb m vecta] = ind2sub(size(results),x(1));
   r(index,:) = [vecta vectb];
   if index >= nb_result | results(x(1)) <= 0
       break;
   end
   index = index+1;
   results(vectb,:,:) = 0;
   results(:,:,vecta) = 0;
end

somme_r = sum((dc1(r(:,1),:)-dc2(r(:,2),:)).*(dc1(r(:,1),:)-dc2(r(:,2),:)),2)
somme_sort = sort(somme_r);
echelle = mean(somme_sort(1:10));
comparaison = 2*echelle:-echelle/nb_result:echelle*((nb_result+1)/nb_result);
somme_r(find((comparaison' - somme_r) > 0))
r = r(find((comparaison' - somme_r) > 0),:);

indices1 = r(:,1);
indices2 = r(:,2);
papp1 = pts1(indices1,:);
papp2 = pts2(indices2,:);
```

Fichier xcorrperiod.m : Corrélation croisée circulaire

```
function [c]=xcorrperiod(a,b)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% "Cross correlation" de deux signaux périodiques de meme taille
%
% Entrées :
%   a, b    : Ensembles de vecteurs ligne de meme taille
%              | Vecteur 1 |
%              | Vecteur 2 |
%              |    ...    |
%              | Vecteur N |
%
% Sortie :
%   c   : Tableau de corrélations (No du vecteur de b * décalage * No du vecteur de a)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if size(a,2) ~= size(b,2)
    error('Les deux vecteurs doivent être de la même taille');
end

% Soustraire les moyennes :
mean_a = zeros(size(a,1),1);
for m = 1:size(a,1)
    ind = isnan(a(m,:));
    if sum(ind(:)) > 0.5*size(a,2)
        disp('a');
        m
        sum(ind(:))
```

```
    end
    mean_a(m) = mean(a(m,find(not(ind))));
end
mean_b = zeros(size(b,1),1);
for m = 1:size(b,1)
    ind = isnan(b(m,:));
    if sum(ind(:)) > 0.5*size(b,2)
        disp('b');
        m
        sum(ind(:))
    end
    mean_b(m) = mean(b(m,find(not(ind))));
end
a(find(isnan(a))) = 0;
b(find(isnan(b))) = 0;
a = a - repmat(mean_a,1,size(a,2));
b = b - repmat(mean_b,1,size(b,2));
c = zeros([size(b,1) size(a,2) size(a,1)]);
% Calculer les corrélations
for n = 1:size(a,1)
    tmp_a = repmat(a(n,:),size(b,1),1);
    for m = 1:size(a,2) % Corrélation de a(n) avec les vecteurs de b
        c(:,m,n) = sum(tmp_a.*[b(:,m:end) b(:,1:m-1)],2);
    end
end
```

Fichier correspondances.m : Vérification de l'appariement par la différence des distances

```
function [ptsa, ptsb, ind1, ind2]=correspondances(ptsa,ptsb,ind1,ind2)
% Calcul des correspondances entre les points en vérifiant les distances
% ptsa : N*3 : point d'intéret a
% ptsb : N*3 : point d'intéret b

s = size(ptsa,1);
mat_dist = zeros(2,s,s);
for m = 1:s
    tmpmat = repmat(ptsa(m,:),s,1)-ptsa;
    mat_dist(1,:,m) = tmpmat(:,1).^2 + tmpmat(:,2).^2 + tmpmat(:,3).^2;
    tmpmat = repmat(ptsb(m,:),s,1)-ptsb;
    mat_dist(2,:,m) = tmpmat(:,1).^2 + tmpmat(:,2).^2 + tmpmat(:,3).^2;
end
repa = reshape(mat_dist(1,:,:),s,s);
repb = reshape(mat_dist(2,:,:),s,s);
dist_ab = abs((repa) - (repb));
dist_ab = dist_ab.*dist_ab
dist_reel = dist_ab(find(dist_ab > 0));
pts_ab = (dist_ab < 0.25*(mean(dist_reel(:))+min(dist_reel(:)))) - eye(s);

result = sum(pts_ab);
% Éliminer ceux qui ne correspondent pas avec les meilleurs
if max(result) > 5
    X = find(result == max(result));
    for m = 1:length(X)
        result(find(pts_ab(:,X(m)) == 0)) == 0;
    end
end
sresult = sort(result);
bon = Inf;
% Déterminer automatiquement le pourcentage de bon points
for m = s:-1:1
    if sresult(m) > (s-m) % Si le nombre de correspondances est plus grand que le nombre
passé avant, ça doit etre bon
```

```
        bon = sresult(m);
    else
        break;
    end
end
bon
indices = find(result >= bon)
result(indices)
ptsa = ptsa(indices,:);
ptsb = ptsb(indices,:);
ind1 = ind1(indices,:);
ind2 = ind2(indices,:);
```

Fichier quaternions.m : Méthode du quaternion dual

```
function [R,t]=quaternions(X,Y)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fonction quaternions
% Trouve le mouvement entre 2 ensembles de points
% X : Ensemble de points d'un mesh (IMPORTANT : Taille N*3)
% Y : Points les plus proches de l'autre mesh correspondant aux points de X
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sx = size(X);
sy = size(Y);
N_1 = 1/sx(1); % 1/N
if sx(2) ~=3 | sy(2) ~= 3
    error('Attention, les vecteurs sont de taille N*3, pas 3*N')
end
if sx(1) ~= sy(1)
    error('Attention, les vecteurs X et Y doivent avoir la meme taille')
end

% Calculs préliminaires des sommes
sumx = sum(X,1);
sumy = sum(Y,1);
x1 = sumx(1);   x2 = sumx(2);   x3 = sumx(3);
y1 = sumy(1);   y2 = sumy(2);   y3 = sumy(3);
x_y = sumx-sumy;
% Calculs préliminaires des multiplications
xy1=sum(X.*Y,1);
xy2=sum(X.*Y(:,[3 1 2]),1);
xy3=sum(X.*Y(:,[2 3 1]),1);
x1y1 = xy1(1);  x1y2 = xy3(1);  x1y3 = xy2(1);
x2y1 = xy2(2);  x2y2 = xy1(2);  x2y3 = xy3(2);
x3y1 = xy3(3);  x3y2 = xy2(3);  x3y3 = xy1(3);

% Calcul de C1= [K(x)K(y)+yx' -y'K(x); -K(y)x y'x];
KyKx = [-x3y3-x2y2 x2y1 x3y1; x1y2 -x3y3-x1y1 x3y2; x1y3 x2y3 -x2y2-x1y1]';
yxT = [x1y1 x1y2 x1y3; x2y1 x2y2 x2y3; x3y1 x3y2 x3y3]';
Kyx = -[x2y3-x3y2 x3y1-x1y3 x1y2-x2y1]';
yTx = x1y1+x2y2+x3y3;
C1(4,4) = yTx;
C1(1:3,1:3) = KyKx + yxT;
C1(1:3,4) = -Kyx;
C1(4,1:3) = -Kyx'; % -yTKx = Kyx'

% Calcul de C2 = [-(K(x) + K(y)) -(x-y); (x-y)' 0];
C2(4,4) = 0;
C2(1:3,1:3) = [0 x3+y3 -(x2+y2);-(x3+y3) 0 x1+y1;x2+y2 -(x1+y1) 0];
C2(1:3,4) = x_y';
C2(4,1:3) = -x_y;

% Calcul de la matrice A = (1/N)*C2'*C2+C1+C1'
[V,D] = eig(N_1*C2'*C2+C1+C1');       % Trouver la plus grande eigenvalue
```

```
q = V(:,find(sum(D,1) == max(D(:))));   % q est le eigenvector correspondant
q = q(:,1); % ligne ajoutée par Vincent

% Calcul de la matrice de rotation
K = [0 -q(3) q(2); q(3) 0 -q(1); -q(2) q(1) 0];
q_u = q(1:3);
R = (q(4)^2 - q_u'*q_u)*eye(3) + 2*q_u*q_u' + 2*q(4)*K; % Matrice de rotation
s = -N_1*C2*q;
W(4,4) = q(4);
W(1:3,1:3) = q(4)*eye(3) - K;
W(1:3,4) = q_u;
W(4,1:3) = -q_u';
p = W'*s;
t = p(1:3); % Vecteur translation
```

Fichier ICP.m : Effectue l'algorithme ICP (ou IPM) + élimination des correspondances

```
function[Rt]= ICP(X,P,portion)
% Fonction qui determine la rotation et la translation Rt afin de recaler
% l'ensemble de point X sur l'ensemble de reference P.
if nargin < 3
    portion = 1;
end

% Initialisation: Rotation identité et translation nulle.
Rt=eye(4);
energie(1)=inf;
eDiff = inf;
i=1;
ind = 1:length(X);
% Début de l'agorithme itératif
while(abs(eDiff) > .01 & i < 500)
    i=i+1;
        % Rotation de l'ensemble de point à recaler
        Xr=X(ind,1:3);
        Xr(:,4)=1;
        Xr=(Rt*Xr');

        % Calcul de l'ensemble de point de P le plus près de Xr
        Y = closestPoints(Xr(1:3,:)',P(:,1:3))'; % Ramener Y dans le bon sens
        Dmin = sqrt(sum((Y - Xr(1:3,:)).^2,1));
    if i == 2 % première itération
         [valeurs,ind] = sort(Dmin);
         length(ind)
         ind2 = ind((round(portion*length(ind))+1):end);
         ind = ind(1:round(portion*length(ind)));
         figure;
         plot3(X(ind,1),X(ind,2),X(ind,3),'.r');
         hold on;
         plot3(X(ind2,1),X(ind2,2),X(ind2,3),'.b');
         hold off;
         Xr = Xr(:,ind); % Prendre une portion des points les plus près
                Y = closestPoints(Xr(1:3,:)',P(:,1:3))'; % Ramener Y dans le bon sens
        Dmin = sqrt(sum((Y - Xr(1:3,:)).^2,1));
    end

        %D=mean(Dmin)/4; % Seuil
    D = Inf;
        Xr=Xr(1:3,find(Dmin<=D)); % Pour ajouter ou enlever le seuil
        Y=Y(1:3,find(Dmin<=D));   % remplacer inf par D ou vice versa
        % Calcul de la rotation/translation résultante.
        [r,t]=quaternions(Xr',Y');
```

```
        rt=[r t ; 0 0 0 1];
        Rt=rt*Rt;    % Application à l'ancienne rotation/translation.
    Xr(4,:)=1;
    Y(4,:)=1;
        energie(i)=mean(sum((Y-rt*Xr).^2,1));
    eDiff = energie(i-1)-energie(i)
end % Fin de l'agorithme ittératif
disp(sprintf('L''algorithme converge en %.0f itérations.',i))
```

Fichier distgeod.h : Calcul des distances géodésiques et des point fingerprints

```
#ifndef _DISTGEOD_H_
#define _DISTGEOD_H_

#include <set>
#include <vector>
#include <MathUtils/Vector3.h>
#include <MathUtils/Matrix4x4.h>
using namespace std;

#ifndef M_PI // Pour les versions bizarres de math.h
#define M_PI        3.14159265358979323846
#define M_PI_2      1.57079632679489661923
#define M_PI_4      0.78539816339744830962
#endif

#define M_2_PI         6.28318530717958647692

#define REEL float         // On pourra verifier si float prends beaucoup moins de temps
apres
#define MAX_REEL FLT_MAX
#define PRECISION 0.00001     // Difference maximale entre 2 angles en radians
#define NB_PTS_MIN 3          // Nombre minimal de points pour calculer la courbe
#define PT_RATIO 0.875               // Au moins 7/8 des points

class ptrayon{
public:
        REEL x;
        REEL y;
        REEL z;
        int rayon;

        ptrayon() {};
        ptrayon(int,REEL,REEL,REEL);
        bool operator<(const ptrayon&) const;
        bool operator==(const ptrayon& other) const
                {return (other.rayon == rayon) &&
                (((other.x - x)*(other.x - x) + (other.y - y)*(other.y - y) + (other.z -
z)*(other.z - z)) < PRECISION*PRECISION);}
};

class triangleVirtuel{
public:
        int ptvoisin1;
        int ptvoisin2;
        int ptvirt;
        CVector3 posvirt;
        triangleVirtuel(int,int,int,CVector3);
};

class triangle{
public:
        int pt1;
```

```
        int pt2;
        int pt3;
        triangle(int,int,int);
        bool operator==(const triangle& other) const
                {return other.pt1 == pt1 && other.pt2 == pt2 && other.pt3 == pt3;}
};

class dist2pt{
public:
        int pt;
        REEL dist;
        dist2pt() {};
        dist2pt(int, REEL);
        bool operator>(const dist2pt&) const;
        bool operator==(const dist2pt& other) const
                {return other.pt == pt && other.dist == other.dist;}
};

class distgeod
{
private :
        enum FastMarchingStates {ALIVE,CLOSE,FAR};

        vector<CVector3> points;      // Ensemble des points (x,y,z)
        vector<triangle> triangles;   // Ensemble des triangles (pt1 pt2, pt3)
        vector<CVector3> normals;     // Ensemble des normales (x,y,z)
        vector<REEL>    rayons;       // Ensemble des rayons voulus
        vector<vector<dist2pt> > geodesics;  // Distances géodésiques
        vector<vector<ptrayon> > fingerprints;       // Ensemble des point fingerprints
        vector<vector<REEL> >  distcentre;   // Distance entre le centre d'un point
fingerprint et le point lui-même
        vector<vector<triangle> > triangleMembership; // Appartenance des triangles pour
chaque point
        vector<vector<triangleVirtuel> > trianglesVirtuels;  // Ensemble des triangles
virtuels créés lors du pré-traitement

        vector<FastMarchingStates> states;
        vector<dist2pt> closeHeap;
        vector<REEL> tmpdists;
        REEL dmax;                                      // Distance géodésique maximale à
atteindre
        REEL seuil;                                     // Distance minimale entre 2 points
choisis
        int nb_ppc;                                     // Nombre de points par cercle
        int nb_pf;                                      // Nombre de points a retourner

        int nbtriobtus;                                 // Nombre d'occurences des triangles
obtus dans les calculs
        int nbtriaigus;                                 // Nombre d'occurences des triangles
aigus dans les calculs
        int nbtrivirts;                                 // Nombre d'occurences des triangles
virtuels dans les calculs
        int nbangle1tropgros;
        int nbangleNtropgros;
        int nbptstroppetit;

        void order(int&, int&, int&);         // Place 3 entiers en ordre
        void compute_geodesic_distances();    // Calcul des distances géodésiques sous
dmax
        void genereTriangleVirtuel(int, int, int);
        void addToFingerPrints(const int&, const int&,const int&);
        void reechantillonne(vector<vector<ptrayon> > &);    // Ré-échantillonne les
courbes
```

```
        void meilleursPF();          // Retourne les meilleurs point fingerprints de
                                                    // l'ensemble reçu en paramètre
selon le critère
                                                    // du plus grand rapport
rayonmax/rayonmin
        void updateHeapStates(int updatedVertexIndex, REEL oldDistance, REEL
newDistance);
        void computeDistOneVertexAlive(int aliveVertexIndex, int toUpdateVertexIndex);
        void computeDistTwoVerticesAlive(int firstAliveVertexIndex, int
secondAliveVertexIndex, int toUpdateVertexIndex);
        int  trouverPointSecondTriangle(int firstAliveVertexIndex, int
secondAliveVertexIndex, int toUpdateVertexIndex);
        void triangleUpdate(const triangle&);
        void projection3D2D();              // Projette les coordonnees 3D sur un plan
        CVector3 ElimineDoublons(int);      // Élimine les doublons et retourne le
centre de gravité
public:
        distgeod(vector<CVector3>, vector<triangle>, vector<CVector3>, vector<REEL>,
REEL, int, int, REEL);
        void getresults(vector<CVector3>&, vector<vector<ptrayon> > &);     // Retour des
résultats
};

#endif
```

Fichier distgeod.cpp : Calcul des distances géodésiques et des point fingerprints

```
#include <algorithm>
#include <cassert>
#include <cfloat>
#include <cmath>
#include <mex.h>
#include <time.h>
#include "mexhelpers.h"
#include "distgeod.h"

#define RAPPORT_MIN 1.1

// devrait etre dans stl... mais visual merde encore et toujours!
#ifdef _MSC_VER
inline REEL min(REEL a, REEL b){
        return a < b ? a : b;
}
#endif

ptrayon::ptrayon(int irayon,REEL ix,REEL iy,REEL iz) {
        rayon = irayon;
        x = ix;
        y = iy;
        z = iz;
}

// Si le meme rayon, comparaison du
// 3e element seulement (habituellement un angle)
bool ptrayon::operator<(const ptrayon& s) const{
        if(rayon < s.rayon) {
                return true;
        } else if(rayon > s.rayon) {
                return false;
        } else { // Si le meme rayon
                if(z < s.z) return true;
                return false;
        }
}
```

```
triangleVirtuel::triangleVirtuel(int ptvo1,int ptvo2,int ptvi,CVector3 pvirt){
        ptvoisin1 = ptvo1;
        ptvoisin2 = ptvo2;
        ptvirt = ptvi;
        posvirt = pvirt;
}

triangle::triangle(int p1,int p2,int p3){
        pt1 = p1;
        pt2 = p2;
        pt3 = p3;
}

dist2pt::dist2pt(int p, REEL d){
        pt = p;
        dist = d;
}

bool dist2pt::operator>(const dist2pt& other) const{
        return dist > other.dist;
}

distgeod::distgeod(vector<CVector3> inipoints, vector<triangle> initriangles,
                                vector<CVector3> ininormals, vector<REEL> inirayons,
REEL inidmax,
                                int ininb_ppc, int ininb_pf, REEL iniseuil) {
        points = inipoints;
        triangles = initriangles;
        normals = ininormals;
        rayons = inirayons;
        nb_ppc = ininb_ppc;
        nb_pf = ininb_pf;
        seuil = iniseuil;
        nbtriobtus = 0;
        nbtriaigus = 0;
        nbtrivirts = 0;
        nbangle1tropgros = 0;
        nbangleNtropgros = 0;
        nbptstroppetit = 0;

        // Mettre les rayons en ordre (juste pour etre sur)
        sort(rayons.begin(), rayons.end());

        dmax = inidmax;
        tmpdists.resize(points.size(),MAX_REEL);
        geodesics.resize(points.size(),vector<dist2pt>());
        fingerprints.resize(points.size(),vector<ptrayon>());
        states.resize(points.size(),FAR);
        trianglesVirtuels.resize(points.size());
}


/**
 *  Lorsque l'item passe de l'etat FAR --> CLOSE il faut l'ajouter au heap
 *  Si celui-ci etait deja dans l'etat CLOSE il faut remettre le heap
 *  a jour afin de selectionner correctement la prochaine valeur acceptee.
 *
 *  Change la valeur des variables membres suivantes: closeHeap et states
 */
void distgeod::updateHeapStates(int updatedVertexIndex, REEL oldDistance, REEL
newDistance){
        assert(states[updatedVertexIndex] == FAR || states[updatedVertexIndex] == CLOSE);
```

```
        if(states[updatedVertexIndex] == FAR) {
                closeHeap.push_back(dist2pt(updatedVertexIndex,newDistance));
                push_heap(closeHeap.begin(), closeHeap.end(),greater<dist2pt>());
                states[updatedVertexIndex] = CLOSE;
        } else {
                // Mettre a jour le heap
                // Le truc poche ici est que l'ordre de l'update en O(n)
                // faudrait se creuser la tete pour faire le tout en O(log(n))...
                vector<dist2pt>::iterator updated_item = find(closeHeap.begin(),
closeHeap.end(),dist2pt(updatedVertexIndex, oldDistance));
                updated_item->dist = newDistance;
                make_heap(closeHeap.begin(), closeHeap.end(),greater<dist2pt>());
        }
}

/**
 *  Met-a-jour les valeurs des distances dans tmpdists lorsque un seul des vertex du
 *  triangle en cours est ALIVE
 */
void distgeod::computeDistOneVertexAlive(int aliveVertexIndex, int toUpdateVertexIndex) {
        bool newDistValue = false;
        assert(states[aliveVertexIndex]==ALIVE);
        assert(states[toUpdateVertexIndex] != ALIVE);

        REEL oldValue = tmpdists[toUpdateVertexIndex];
        REEL d12 = (points[aliveVertexIndex] - points[toUpdateVertexIndex]).Norm();
        tmpdists[toUpdateVertexIndex] = min(tmpdists[toUpdateVertexIndex],tmpdists
[aliveVertexIndex] + d12);

        if(oldValue > tmpdists[toUpdateVertexIndex])
                updateHeapStates(toUpdateVertexIndex,oldValue,tmpdists
[toUpdateVertexIndex]);
}

// Trouve le point manquant du triangle face à l'angle obtu
int distgeod::trouverPointSecondTriangle(int vertexA, int vertexB, int vertex_obtu) {
        int i,j,k;
        bool wrongtriangle = true;
        for(i = 0; i < triangleMembership[vertexA].size(); i++) {
                for(j = 0; j < triangleMembership[vertexB].size(); j++) {
                        if(triangleMembership[vertexA][i] == triangleMembership[vertexB]
[j]) {
                                wrongtriangle = false;
                                for(k = 0; k < triangleMembership[vertex_obtu].size();
        k++) {
                                        if(triangleMembership[vertex_obtu][k] ==
triangleMembership[vertexA][i]) {
                                                wrongtriangle = true;
                                                break;
                                        }
                                }
                        }
                        if(!wrongtriangle) break;
                }
                if(!wrongtriangle) break;
        }
        if(!wrongtriangle) {
                if(triangleMembership[vertexA][i].pt1 != vertexA &&
                   triangleMembership[vertexA][i].pt1 != vertexB &&
                   triangleMembership[vertexA][i].pt1 != vertex_obtu) {
                        return triangleMembership[vertexA][i].pt1;
                }
                else if(triangleMembership[vertexA][i].pt2 != vertexA &&
```

```
                                triangleMembership[vertexA][i].pt2 != vertexB &&
                                triangleMembership[vertexA][i].pt2 != vertex_obtu) {
                        return triangleMembership[vertexA][i].pt2;
                }
                else if(triangleMembership[vertexA][i].pt3 != vertexA &&
                                triangleMembership[vertexA][i].pt3 != vertexB &&
                                triangleMembership[vertexA][i].pt3 != vertex_obtu){
                        return triangleMembership[vertexA][i].pt3;
                }
        }
        return -1;
}

/**
 *  Met-a-jour les valeurs des distances dans tmpdists lorsque deux des vertex du
 *  triangle en cours est ALIVE.  Pour l'origine des equations, consulter l'article de
 *  Sethian et Kimmel.
 */

void distgeod::computeDistTwoVerticesAlive(int firstAliveVertexIndex, int
secondAliveVertexIndex, int toUpdateVertexIndex) {
        CVector3 AC = points[firstAliveVertexIndex] - points[toUpdateVertexIndex];
        CVector3 BC = points[secondAliveVertexIndex] - points[toUpdateVertexIndex];
        REEL a = BC.Norm();
        REEL b = AC.Norm();
        REEL u;

        REEL oldValue = tmpdists[toUpdateVertexIndex];
        tmpdists[toUpdateVertexIndex] = min(tmpdists[toUpdateVertexIndex], tmpdists
[firstAliveVertexIndex] + b);
        tmpdists[toUpdateVertexIndex] = min(tmpdists[toUpdateVertexIndex], tmpdists
[secondAliveVertexIndex] + a);

        // Vérifier que le vertex à mettre à jour n'est pas dans un angle obtu
        if((points[toUpdateVertexIndex]-points[firstAliveVertexIndex])*
           (points[toUpdateVertexIndex]-points[secondAliveVertexIndex]) >= 0) {
                // calcul quadratique puisque non obtu
                REEL cTheta = AC.Normalize()*BC.Normalize();
                REEL theta = acos(cTheta);
                REEL sTheta = sin(theta);

                u = tmpdists[secondAliveVertexIndex] - tmpdists[firstAliveVertexIndex];

                REEL A = a*a + b*b - 2*a*b*cTheta;
                REEL B = 2*b*u*(a*cTheta - b);
                REEL C = b*b*(u*u - a*a*sTheta*sTheta);
                REEL delta = B*B - 4*A*C;

                assert(delta>0);

                REEL sdelta = sqrt(delta);

                REEL t1 = (-B + sdelta)/(2*A);
                REEL t2 = (-B - sdelta)/(2*A);

                if(u < t1 && a*cTheta < b*(t1-u)/t1  && b*(t1-u)/t1  < a/cTheta )
                        tmpdists[toUpdateVertexIndex] = min(tmpdists[toUpdateVertexIndex],
tmpdists[firstAliveVertexIndex] + t1);
                if(u < t2 && a*cTheta < b*(t2-u)/t2  && b*(t2-u)/t2  < a/cTheta )
                        tmpdists[toUpdateVertexIndex] = min(tmpdists[toUpdateVertexIndex],
tmpdists[firstAliveVertexIndex] + t2);
                nbtriaigus++;
        }
```

```
        else { // Le triangle est obtu pour le point à mettre à jour
                // Vérifier les triangles virtuels.
                bool fini = false;
                for(vector<triangleVirtuel>::iterator t = trianglesVirtuels
[toUpdateVertexIndex].begin(); t != trianglesVirtuels[toUpdateVertexIndex].end() && !
fini; t++) {
                        if((t->ptvoisin1 == firstAliveVertexIndex && t->ptvoisin2 ==
secondAliveVertexIndex) ||
                           (t->ptvoisin2 == firstAliveVertexIndex && t->ptvoisin1 ==
secondAliveVertexIndex)){ // Si les 2 voisins correspondent
                                AC = t->posvirt - points[toUpdateVertexIndex];
                                b = AC.Norm();
                                tmpdists[toUpdateVertexIndex] = min(tmpdists
[toUpdateVertexIndex], tmpdists[t->ptvirt] + b);
                                for(int i = 0; i < 2; i++) { // Essayer les 2 possibilités
                                        if(i == 0) { // D'abord le premier vertex devient B
                                                BC = points[firstAliveVertexIndex] - points
[toUpdateVertexIndex];
                                                a = BC.Norm();
                                                u = tmpdists[firstAliveVertexIndex] -
tmpdists[t->ptvirt];
                                        } else { // Ensuite le second vertex devient B
                                                BC = points[secondAliveVertexIndex] - points
[toUpdateVertexIndex];
                                                a = BC.Norm();
                                                u = tmpdists[secondAliveVertexIndex] -
tmpdists[t->ptvirt];
                                        }
                                        REEL cTheta = AC.Normalize()*BC.Normalize();
                                        REEL theta = acos(cTheta);
                                        REEL sTheta = sin(theta);

                                        REEL A = a*a + b*b - 2*a*b*cTheta;
                                        REEL B = 2*b*u*(a*cTheta - b);
                                        REEL C = b*b*(u*u - a*a*sTheta*sTheta);
                                        REEL delta = B*B - 4*A*C;

                                        assert(delta>0);

                                        REEL sdelta = sqrt(delta);

                                        REEL t1 = (-B + sdelta)/(2*A);
                                        REEL t2 = (-B - sdelta)/(2*A);

                                        if(u < t1 && a*cTheta < b*(t1-u)/t1  && b*(t1-u)/t1
< a/cTheta )
                                                tmpdists[toUpdateVertexIndex] = min(tmpdists
[toUpdateVertexIndex], tmpdists[t->ptvirt] + t1);
                                        if(u < t2 && a*cTheta < b*(t2-u)/t2  && b*(t2-u)/t2
< a/cTheta )
                                                tmpdists[toUpdateVertexIndex] = min(tmpdists
[toUpdateVertexIndex], tmpdists[t->ptvirt] + t2);
                                }
                                fini = true;
                                nbtrivirts++;
                        }
                }
                nbtriobtus++;
        }

        if(oldValue > tmpdists[toUpdateVertexIndex])
                updateHeapStates(toUpdateVertexIndex,oldValue,tmpdists
[toUpdateVertexIndex]);
```

```
}

/**
 * En se basant sur les distances deja calcule et sur l'etat des points (ALIVE,
 * CLOSE,FAR) recalculer le meilleur estimé possible de la distance géodésique.
 *
 * Pour les details mathematique se referer a "Fast Marching Methods on
 * Triangulated Domains" de Kimmel et Sethian
 *
 * Input: Le triangle sur lequel on desire calcul un update
 * Output: Mise-a-jour de la variable membre tmpdists et states
 */
void distgeod::triangleUpdate(const triangle& triangleToUpdate) {

        int nb_alive=0;
        if(states[triangleToUpdate.pt1] == ALIVE) nb_alive++;
        if(states[triangleToUpdate.pt2] == ALIVE) nb_alive++;
    if(states[triangleToUpdate.pt3] == ALIVE) nb_alive++;

        assert(nb_alive >= 1);

        if(nb_alive==1){
                if(states[triangleToUpdate.pt1] == ALIVE) {
                        computeDistOneVertexAlive(triangleToUpdate.pt1,
triangleToUpdate.pt2);
                        computeDistOneVertexAlive(triangleToUpdate.pt1,
triangleToUpdate.pt3);
                } else if(states[triangleToUpdate.pt2] == ALIVE) {
                        computeDistOneVertexAlive(triangleToUpdate.pt2,
triangleToUpdate.pt1);
                        computeDistOneVertexAlive(triangleToUpdate.pt2,
triangleToUpdate.pt3);
                } else if(states[triangleToUpdate.pt3] == ALIVE) {
                        computeDistOneVertexAlive(triangleToUpdate.pt3,
triangleToUpdate.pt1);
                        computeDistOneVertexAlive(triangleToUpdate.pt3,
triangleToUpdate.pt2);
                }

        } else if(nb_alive == 2) {
                if(states[triangleToUpdate.pt1] == CLOSE)
                        computeDistTwoVerticesAlive
(triangleToUpdate.pt2,triangleToUpdate.pt3, triangleToUpdate.pt1);
                else if(states[triangleToUpdate.pt2] == CLOSE)
                        computeDistTwoVerticesAlive
(triangleToUpdate.pt1,triangleToUpdate.pt3, triangleToUpdate.pt2);
                else if(states[triangleToUpdate.pt3] == CLOSE)
                        computeDistTwoVerticesAlive
(triangleToUpdate.pt1,triangleToUpdate.pt2, triangleToUpdate.pt3);
        }
}

void distgeod::genereTriangleVirtuel(int vertex_obtu, int vertexA, int vertexB){
        int vA = vertexA, vB = vertexB, vOut = vertex_obtu;
        CVector3 posvirt; // Position virtuelle du point trouvé
        CVector3 ABtmp, ACtmp, BCtmp, ADtmp;
        CMatrix4x4 Mglobal, Mrotate, Morigin, Mpoint;

        // Trouver le triangle utilisant les 2 vertex non obtus
        int indice = trouverPointSecondTriangle(vA, vB, vOut);
        if(indice != -1) { // Si le triangle existe
                // Effectuer une rotation autour de l'axe AB de l'angle entre
                // la normale du triangle courant et l'angle du triangle trouvé
```

```
                CVector3 AB = points[vA] - points[vB];
                CVector3 AC = points[vA] - points[vOut];
                CVector3 BC = points[vB] - points[vOut];
                CVector3 AD = points[vA] - points[indice];
                CVector3 n1 = (AC^AB).Normalize(); // Normale au triangle 1
                CVector3 n2 = (AB^AD).Normalize(); // Normale au triangle 2
                CVector3 axe_rot = (n1^n2).Normalize(); // Axe de rotation (PAS la même
chose que AB.Normalize())
                REEL cTheta = n1*n2;
                REEL theta = acos(cTheta); // Angle entre les 2 normales
                REEL sTheta = sin(theta);

                // Pour la formule, voir http://mathworld.wolfram.com/RotationFormula.html
                // Pour la forme matricielle, voir
http://mathworld.wolfram.com/RodriguesRotationFormula.html
                // Prenons le point A comme origine pour la rotation :
                Mpoint = Mrotate = Morigin = CMatrix4x4::I;
                Morigin[3] = -points[vA].x;          // Pour l'origine de l'axe de
rotation
                Morigin[7] = -points[vA].y;
                Morigin[11] = -points[vA].z;
                Mpoint[3] = points[indice].x; // Point à rotater
                Mpoint[7] = points[indice].y;
                Mpoint[11] = points[indice].z;
                Mrotate.Rotate(axe_rot, theta);      // Entrer la rotation
                Mrotate[3] = points[vA].x;           // Ajouter la translation
                Mrotate[7] = points[vA].y;
                Mrotate[11] = points[vA].z;
                Mglobal = Morigin*Mrotate;           // Conserver la transformation
globale
                Mpoint *= Mglobal;                          // Effectuer la rotation
                posvirt = CVector3(Mpoint[3],Mpoint[7],Mpoint[11]);

        // Validation des résultats
                CVector3 posvirtcheck = points[vA] - (AD*cTheta + axe_rot*(axe_rot*AD)*(1-
cTheta) + (AD^axe_rot)*sTheta);
                if((posvirt - posvirtcheck).Norm() > 0.1) {
                        mexWarnMsgTxt("Erreur de calcul potentielle !");
                        mexPrintf("[%f %f %f] != [%f %f %f]
\n",posvirt.x,posvirt.y,posvirt.z,posvirtcheck.x,posvirtcheck.y,posvirtcheck.z);
                        mexPrintf("D = [%f %f %f] axe_rot = [%f %f %f] theta = %f
\n",points[indice].x,points[indice].y,points[indice].
z,axe_rot.x,axe_rot.y,axe_rot.z,theta);
                        mexPrintf("Morigin = [ %f %f %f %f ; \n  %f %f %f %f ; \n  %f %f %
f %f ; \n %f %f %f %f ] \n",
                                        Morigin[0],Morigin[1],Morigin[2],Morigin[3],
                                        Morigin[4],Morigin[5],Morigin[6],Morigin[7],
                                        Morigin[8],Morigin[9],Morigin[10],Morigin[11],
                                        Morigin[12],Morigin[13],Morigin[14],Morigin[15]);
                        mexPrintf("Mrotate = [ %f %f %f %f ; \n  %f %f %f %f ; \n  %f %f %
f %f ; \n %f %f %f %f ] \n",
                                        Mrotate[0],Mrotate[1],Mrotate[2],Mrotate[3],
                                        Mrotate[4],Mrotate[5],Mrotate[6],Mrotate[7],
                                        Mrotate[8],Mrotate[9],Mrotate[10],Mrotate[11],
                                        Mrotate[12],Mrotate[13],Mrotate[14],Mrotate[15]);
                        mexPrintf("Mglobal = [ %f %f %f %f ; \n  %f %f %f %f ; \n  %f %f %
f %f ; \n %f %f %f %f ] \n",
                                        Mglobal[0],Mglobal[1],Mglobal[2],Mglobal[3],
                                        Mglobal[4],Mglobal[5],Mglobal[6],Mglobal[7],
                                        Mglobal[8],Mglobal[9],Mglobal[10],Mglobal[11],
                                        Mglobal[12],Mglobal[13],Mglobal[14],Mglobal[15]);
                }
        // Fin de la validation
```

```
                CVector3 CD = points[vertex_obtu] - posvirt;

        // Validation des résultats
                CVector3 nchk1 = (AC^CD).Normalize(); // Normale du triangle virtuel
                CVector3 nchk2 = (AC^AB).Normalize(); // Normale du triangle obtu
                if((nchk1 - nchk2).Norm() > 0.1 && (nchk1 + nchk2).Norm() > 0.1) {
                        mexWarnMsgTxt("Premier : Erreur de calcul potentielle !");
                        mexPrintf("(n1 =[%f %f %f]) != (n2 = [%f %f %f])
\n",nchk1.x,nchk1.y,nchk1.z,nchk2.x,nchk2.y,nchk2.z);
                }
        // Fin de la validation

                // TANT QUE le point calculé après rotation n'est pas valide selon les 2
angles
                int n = 1;
                while(indice != -1 && ((CD*-AC) < 0 || (CD*-BC) < 0)) {
                        // Trouver entre quels points du dernier triangle la zone valide
passe
                        if((CD*-AC) < 0) { // Les point valides sont D et A
                                vOut = vB;              // B devient le point hors du
triangle courant
                                vB = indice;    // Le dernier point trouvé devient B
                        } else if((CD*-BC) < 0) { // Les point valides sont D et B
                                vOut = vA;              // A devient le point hors du
triangle courant
                                vA = indice;    // Le dernier point trouvé devient A
                        }
                        indice = trouverPointSecondTriangle(vA, vB, vOut);
                        if(indice != -1) {
                                ABtmp = points[vA] - points[vB];
                                ACtmp = points[vA] - points[vOut];
                                BCtmp = points[vB] - points[vOut];
                                ADtmp = points[vA] - points[indice];

                                n1 = (ACtmp^ABtmp).Normalize(); // Normale au triangle 1
                                n2 = (ABtmp^ADtmp).Normalize(); // Normale au triangle 2
                                axe_rot = (n1^n2).Normalize(); // Axe de rotation (PAS la
même chose que AB.Normalize())
                                cTheta = n1*n2;
                                theta = acos(cTheta); // Angle entre les 2 normales

                                Mpoint = Mrotate = Morigin = CMatrix4x4::I;
                                Morigin[3] = -points[vA].x;             // Pour l'origine de
l'axe de rotation
                                Morigin[7] = -points[vA].y;
                                Morigin[11] = -points[vA].z;
                                Mpoint[3] = points[indice].x; // Point à rotater
                                Mpoint[7] = points[indice].y;
                                Mpoint[11] = points[indice].z;
                                Mrotate.Rotate(axe_rot, theta);         // Entrer la rotation
                                Mrotate[3] = points[vA].x;              // Ajouter la
translation
                                Mrotate[7] = points[vA].y;
                                Mrotate[11] = points[vA].z;
                                Mglobal = Morigin*Mrotate*Mglobal;      // Conserver la
transformation globale
                                Mpoint *= Mglobal;                              // Effectuer
la rotation
                                posvirt = CVector3(Mpoint[3],Mpoint[7],Mpoint[11]);

                                CD = points[vertex_obtu] - posvirt;
                                if(CD.Norm() > 2*dmax) { // Inutile de continuer après
2*dmax
```

```
                                        //mexWarnMsgTxt("On dépasse !");
                                        indice = -1;
                                }
                                else {
                        // Validation des résultats
                                        nchk1 = (AC^CD).Normalize(); // Normale du triangle
virtuel
                                        nchk2 = (AC^AB).Normalize(); // Normale du triangle
obtu
                                        if((nchk1 - nchk2).Norm() > 0.1 && (nchk1 + nchk2).
Norm() > 0.1) {
                                                mexWarnMsgTxt("Boucle : Erreur de calcul
potentielle !");
                                                mexPrintf("%i : (n1 =[%f %f %f]) != (n2 = [%
f %f %f]) \n",n,nchk1.x,nchk1.y,nchk1.z,nchk2.x,nchk2.y,nchk2.z);
                                        }
                        // Fin de la validation
                                }
                                n++;
                        }
                }
        }

        if(indice != -1) {
                trianglesVirtuels[vertex_obtu].push_back(triangleVirtuel
(vertexA,vertexB,indice,posvirt));
                //mexPrintf("Nouveau triangle virtuel : %i %i %i %
i\n",vertex_obtu,vertexA,vertexB,indice);
        }
}

// Calcul des distances géodésiques sous dmax
void distgeod::compute_geodesic_distances() {
        // calcule des distances entre voisins immediats
        // et calcule de l'ensemble des triangles auxquels chaque points participe
        triangleMembership.resize(points.size());
        for(vector<triangle>::iterator t = triangles.begin(); t != triangles.end(); t++)
{
                triangleMembership[t->pt1].push_back(*t);
                triangleMembership[t->pt2].push_back(*t);
                triangleMembership[t->pt3].push_back(*t);
        }

        for(t = triangles.begin(); t != triangles.end(); t++) {
                if((points[t->pt1]-points[t->pt2])*(points[t->pt1]-points[t->pt3])<0){
                        genereTriangleVirtuel(t->pt1,t->pt2,t->pt3); // Angle obtu en pt1
                }
                if((points[t->pt2]-points[t->pt1])*(points[t->pt2]-points[t->pt3])<0){
                        genereTriangleVirtuel(t->pt2,t->pt1,t->pt3); // Angle obtu en pt2
                }
                if((points[t->pt3]-points[t->pt1])*(points[t->pt3]-points[t->pt2])<0){
                        genereTriangleVirtuel(t->pt3,t->pt1,t->pt2); // Angle obtu en pt3
                }
        }

        // Calcul des distances géodésiques pour tous les points
        for(int i = 0; i < points.size(); i++) {
                fill(states.begin(),states.end(),FAR);
                fill(tmpdists.begin(),tmpdists.end(),MAX_REEL);

                dist2pt trial(i,0.0);
                tmpdists[i] = 0;
                closeHeap.clear();
```

```
                closeHeap.push_back(trial);

                // Boucler jusqu'à ce que la distance maximale soit atteinte
                while(trial.dist < dmax && !closeHeap.empty() ) {

                        // Prendre le premier du heap (le close avec la distance minimale)
                        trial = closeHeap[0];
                        pop_heap(closeHeap.begin(), closeHeap.end(),greater<dist2pt>
()); // Retirer le premier element
                        closeHeap.pop_back();

                        states[trial.pt] = ALIVE;      // Le point passe à l'état ALIVE
                        tmpdists[trial.pt] = trial.dist; // Conserver la distance pour le
calcul des rayons
                        // Ajout aux distances geodesiques
                        if(i < trial.pt) geodesics[i].push_back(trial); // Eviter
d'ajouter 2 fois

                        // Vérifier les voisins du point trial
                        for(vector<triangle>::iterator neigh_tri_it = triangleMembership
[trial.pt].begin() ;
                                neigh_tri_it != triangleMembership[trial.pt].end() ;
neigh_tri_it++) {

                                triangleUpdate(*neigh_tri_it);

                                // Meme si sur nom pourrait laisser croire que la fonction
ne genere que des distances,
                                // elle calcule aussi des cercles geodesique (courbe pour
surlaquelle la distance
                                // geodesique a un point est constante)
                                // Dans le cas ou il neighbors est ALIVE, il faut verifier
ces courbes
                                addToFingerPrints(i,neigh_tri_it->pt1,trial.pt);
                                addToFingerPrints(i,neigh_tri_it->pt2,trial.pt);
                                addToFingerPrints(i,neigh_tri_it->pt3,trial.pt);
                        }
                }
                closeHeap.clear(); // Effacer tous les elements
        }
        triangleMembership.clear();
}

void distgeod::addToFingerPrints(const int& origin, const int& vertexIndex, const int&
trialVertexIndex){
        int a,b,k;
        CVector3 interpole;

        if(states[vertexIndex] == ALIVE  && vertexIndex != trialVertexIndex)   {
                if(tmpdists[vertexIndex] <= 0.0 && vertexIndex != origin) { // Ne devrait
jamais arriver
                        mexWarnMsgTxt("Il manque une distance geodesique !");
                } else {
                        // Pour tous les rayons reçus en paramètre, vérifier si le segment
                        // situé entre le point trial (maintenant à l'état ALIVE) et un de
                        // ses voisins aussi dans l'état ALIVE contient un point dont la
                        // distance se trouve à l'un des rayons voulus.

                        for(k = 0; k < rayons.size(); k++) {
                                // Si on dépasse la distance courante,
                                // il est impossible d'y trouver un rayon
                                if(tmpdists[trialVertexIndex] < rayons[k]) break;
                                a = b = -1;
```

```
                                if(tmpdists[vertexIndex] < rayons[k]) { a = vertexIndex; b
= trialVertexIndex; }
                                if(a < 0) continue;
                                interpole = 1.0/(tmpdists[b] - tmpdists[a]) *
                                        ((tmpdists[b] - rayons[k])*points[a]
                                        + (rayons[k] - tmpdists[a])*points[b]) - points
[origin];
                                fingerprints[origin].push_back(ptrayon
(k,interpole.x,interpole.y,interpole.z));
                        }
                }
        }
}

// Élimine les doublons et retourne le centre de gravité
CVector3 distgeod::ElimineDoublons(int i){
        int j,k,n;
        bool doublon;
        CVector3 centre(0.0,0.0,0.0);
        vector<ptrayon> xyz;
        xyz.clear();
        n = 0;
        for(j = 0; j < fingerprints[i].size(); j++) {
                doublon = false;
                for(k = 0; k < xyz.size();k++) {
                        if(fingerprints[i][j] == xyz[k]) doublon = true;
                }
                if(doublon) {
                        fingerprints[i][j].rayon = -1; // Identifier les doublons
                }
                else {
                        xyz.push_back(fingerprints[i][j]);
                        centre += CVector3(fingerprints[i][j].x, fingerprints[i][j].y,
fingerprints[i][j].z);
                        n++;
                }
        }
        for(j = 0; j < fingerprints[i].size(); j++) {
                if(fingerprints[i][j].rayon == -1) {
                        fingerprints[i].erase(&fingerprints[i][j]);
                        j--;
                }
        }
        return centre.Normalize();
}

// Fingerprints 3D -> 2D
// Change le contenu de fingerprints initialement : coordonnées x,y,z
// En une projection sur le plan normal au point  : coordonnées x,y,dist
void distgeod::projection3D2D() {
        int i,j;
        REEL x,y,angle;
        vector<REEL> sumx,sumy,nbpt;
        vector<ptrayon> xyz;
        CVector3 base1, base2, pointcur, axe;
        sumx.resize(rayons.size());
        sumy.resize(rayons.size());
        nbpt.resize(rayons.size());
        distcentre.resize(fingerprints.size());
        for(i = 0; i < fingerprints.size(); i++) {
                if(fingerprints[i].size() < 1) continue;
                // Calcul de la base
                axe = ElimineDoublons(i);
```

```
                pointcur = CVector3(fingerprints[i][0].x,fingerprints[i][0].y,fingerprints
[i][0].z);
                base1 = normals[i]^pointcur;  // Produit vectoriel
                base1.Normalize();                              // Normalisation
                base2 = base1^normals[i];               // Produit vectoriel
                base2.Normalize();                              // Normalisation

                // Boucle de calcul des angles et taille des rayons
                xyz.clear();
                fill(sumx.begin(),sumx.end(),0.0);
                fill(sumy.begin(),sumy.end(),0.0);
                fill(nbpt.begin(),nbpt.end(),0.0);
                for(j = 0; j < fingerprints[i].size(); j++) {
                        // Projection sur le plan normal au point i
                        pointcur = CVector3(fingerprints[i][j].x,fingerprints[i][j].
y,fingerprints[i][j].z);
                        sumx[fingerprints[i][j].rayon] += (fingerprints[i][j].x = base1 *
pointcur);
                        sumy[fingerprints[i][j].rayon] += (fingerprints[i][j].y = base2 *
pointcur);
                        nbpt[fingerprints[i][j].rayon]++;
                }
                // Trouver le centre du fingerprint
                distcentre[i].resize(rayons.size());
                for(j = 0; j < rayons.size(); j++) {
                        sumx[j] /= nbpt[j];
                        sumy[j] /= nbpt[j];
                        distcentre[i][j] = sqrt(sumx[j]*sumx[j] + sumy[j]*sumy[j]);
                }
                for(j = 0; j < fingerprints[i].size(); j++) {
                        x = (fingerprints[i][j].x -= sumx[fingerprints[i][j].rayon]); //
Recentrer x
                        y = (fingerprints[i][j].y -= sumy[fingerprints[i][j].rayon]); //
Recentrer y
                        if(fabs(x) < PRECISION) {
                                if(y < 0.0) { angle = M_PI+M_PI_2; } // 3 PI / 2
                                else { angle = M_PI_2; }                                //
PI / 2
                        }
                        else {
                                angle = atan(y/x);
                                if(x < 0) angle += M_PI;                // Pour 360 degres (2
PI)
                                if(angle < 0) angle += M_2_PI;          // Toujours positif
                        }
                        fingerprints[i][j].z = angle;
                }
        }
}

// Refait un echantillonnage regulier
void distgeod::reechantillonne(vector<vector<ptrayon> > &resultsPF) {
        int i; //,j,k;
        //ptrayon pr;

        vector<vector<ptrayon> > pf;
        vector<vector<ptrayon> > courbes;
        vector<ptrayon> ctmp;
        for(i = 0; i < fingerprints.size(); i++) pf.push_back(ctmp);
        for(i = 0; i < rayons.size(); i++) courbes.push_back(ctmp);

/*      int index;
        REEL x,y,angle,anglea,angleb,inc,zonevide,zonetmp;
```

```
	inc = M_2_PI/nb_ppc; // Increment en radians pour le cercle

	for(i = 0; i < fingerprints.size(); i++) {
		pf[i].clear(); // S'assurer de partir avec une structure vide
		if(fingerprints[i].size() == 0) continue; // Aucun intérêt à regarder un
point sans courbe
		// Effaces les courbes précédentes
		for(j = 0; j < rayons.size(); j++)    courbes[j].clear();

		// Boucle de calcul des angles et taille des rayons
		for(j = 0; j < fingerprints[i].size(); j++) {
			// Projection sur le plan normal au point i
			x = fingerprints[i][j].x;
			y = fingerprints[i][j].y;
			if(fabs(x) < PRECISION) {
				if(y < 0.0) { angle = M_PI+M_PI_2; }
				else { angle = M_PI_2; }
			}
			else {
				angle = atan(y/x);
				if(x < 0) angle += M_PI;              // Pour 360 degres (2
PI)
				if(angle < 0) angle += M_2_PI;        // Toujours positif
			}
			// Calcul de la longueur du rayon
			x = sqrt(fingerprints[i][j].z);
			// Ajouter le tout à la liste
			courbes[fingerprints[i][j].rayon].push_back(ptrayon(fingerprints
[i][j].rayon,x,0,angle));
		}

		angle = 0.0;
		for(j = 0; j < rayons.size(); j++) {
			if(courbes[j].size() <= 2) continue; // Sécurité seulement
			// Mettre les points de la courbe en ordre
			sort(courbes[j].begin(), courbes[j].end());
			// Trouver la plus grande zone sans points de manière
			// à former le plus petit arc de cercle possible
			index = 0;
			zonevide = M_2_PI + (courbes[j][0].z - courbes[j][courbes[j].size
()-1].z);
			for(k = 1; k < courbes[j].size(); k++) {
				zonetmp = courbes[j][k].z - courbes[j][k-1].z;
				if(zonetmp > zonevide) { index = k;   zonevide = zonetmp; }
			}
			// Soustraire le premier angle pour que le premier devienne 0
			for(k = 1; k < courbes[j].size(); k++) {
				courbes[j][k].z -= courbes[j][index].z;
				if(courbes[j][k].z < 0) courbes[j][k].z += M_2_PI;
			}
			sort(courbes[j].begin(), courbes[j].end());
			// Ajouter le dernier element au debut avec
			// un angle negatif pour completer la boucle
			pr = courbes[j].back();
			angle += pr.z; // Calcul du total des angles des cercles
			// Pour que le premier angle soit négatif et le second positif
			// (donc pour que le 0 soit inclus dans l'intervalle)
			pr.z -= M_2_PI;
			courbes[j].insert(courbes[j].begin(), pr); // Insere au debut
			// Ajouter le premier element + 2 pi à la fin
			pr = courbes[j][1];
			pr.z += M_2_PI;
			courbes[j].push_back(pr); // Insere à la fin
```

```
                }
                // Au moins un certain nombre de points à interpoler
                //if(angle < (PT_RATIO * M_2_PI * rayons.size())) continue;

                // Re-echantillonnage en nb_ppc (nombre de points par cercle)
                for(j = 0; j < rayons.size(); j++) {
                        index = 0;
                        // Prendre tous les angles por formet un cercle
                        for(angle = 0.0; angle < (M_2_PI-0.5*inc); angle += inc) {
                                x = -1.0;
                                while(index+1 < courbes[j].size()) {
                                        anglea = courbes[j][index].z;
                                        angleb = courbes[j][index + 1].z;
                                        if(anglea < angle && angleb >= angle) {
                                                // Interpolation linéaire entre les 2 rayons
                                                x = ((angleb - angle)*courbes[j][index].x
                                                        +(angle - anglea)*courbes[j][index +
1].x)
                                                        /(angleb - anglea);
                                                break;
                                        }
                                        else index++;
                                }
                                if(x < 0) {
                                        mexWarnMsgTxt("Un angle n'a pas été trouvé");
                                        break;
                                }
                                // Ici : x = rayon interpolé
                                //              y = no du point [1,...,nb_ppc]
                                //              angle = [0,inc,2*inc,...,2*PI[
                                pf[i].push_back(ptrayon(j,i,x,0.0));
                        }
                }
        }*/
        resultsPF = pf;
}

// Choix des meilleurs point fingerprints
// (Élimination des moins bons)
void distgeod::meilleursPF(){
        bool valide;
        int i,j,k,nbpts,ptcur;
        REEL ratio, tmprayon, tmpangle;
        vector<REEL> rayonmin, rayonmax;
        vector<vector<ptrayon> > fingerprint;
        vector<dist2pt> raymaxmin;
        vector<bool> isaccepte;
        vector<vector<ptrayon> > pf_to_keep;
        rayonmin.resize(rayons.size());
        rayonmax.resize(rayons.size());
        fingerprint.resize(rayons.size());
        for(i = 0; i < fingerprints.size(); i++) {
                valide = true;
                for(j = 0; j < fingerprints[i].size(); j++) {
                        fingerprint[fingerprints[i][j].rayon].push_back(fingerprints[i]
[j]);
                }
                for(j = 0; j < fingerprint.size() && valide ; j++) {
                        if(fingerprint[j].size() > NB_PTS_MIN) {
                                sort(fingerprint[j].begin(), fingerprint[j].end()); //
Mettre en ordre
                                // Plus le rayon est grand, plus il est anormal qu'il en
manque une partie
```

```
                                // Pour le premier rayon, on accepte PI (1/2 tour), mais
pour le dernier, on n'accepte que PI/2 (1/4 de tour)
                                // Si un fingerprint est décentré au point de se trouver
complètement à l'extérieur du point central,
                                // le dernier rayon devrait rater au moins un quart de
cercle et sera alors éliminé.
                                if(rayons.size() > 1) {
                                        tmpangle = M_PI_2 + ((M_PI_2*(rayons.size() - 1 -
j))/(rayons.size() - 1));
                                } else {
                                        tmpangle = M_PI;
                                }
                                for(k = 0; k < fingerprint[j].size() && valide; k++) {
                                        if(k == 0) {
                                                if(fabs(fingerprint[j][k].z - fingerprint
[j].back().z + M_2_PI) >= (tmpangle - PRECISION)) {
                                                        valide = false;
                                                        nbangle1tropgros++;
                                                }
                                        } else {
                                                if(fabs(fingerprint[j][k].z - fingerprint[j]
[k-1].z) >= (tmpangle - PRECISION)) {
                                                        valide = false;
                                                        nbangleNtropgros++;
                                                }
                                        }
                                }
                        } else {
                                valide = false;
                                nbptstroppetit++;
                        }
                }
                for(j = 0; j < fingerprint.size(); j++) {
                        fingerprint[j].clear();
                }
                if(!valide) continue; // Passer au point suivant, le point courant n'a pas
assez de points

                // Prendre le plus grand rapport rayon max / rayon min pour le point
courant
                fill(rayonmax.begin(),rayonmax.end(),0.0);
                fill(rayonmin.begin(),rayonmin.end(),MAX_REEL);
                for(j = 0; j < fingerprints[i].size(); j++) {
                        tmprayon = fingerprints[i][j].x*fingerprints[i][j].x +
fingerprints[i][j].y*fingerprints[i][j].y;
                        if(tmprayon > rayonmax[fingerprints[i][j].rayon])
                                rayonmax[fingerprints[i][j].rayon] = tmprayon;
                        if(tmprayon < rayonmin[fingerprints[i][j].rayon])
                                rayonmin[fingerprints[i][j].rayon] = tmprayon;
                }
                ratio = 0;
                for(j = 0; j < rayons.size(); j++) {
                        if(rayonmax[j]/rayonmin[j] > ratio)
                                ratio = rayonmax[j]/rayonmin[j];
                }
                raymaxmin.push_back(dist2pt(i,ratio));
        }
        isaccepte.clear();
        isaccepte.resize(points.size(),false);
        if(raymaxmin.size() > 0) {
                // Faire en sorte que les meilleurs points soient au début du vecteur
                sort(raymaxmin.begin(), raymaxmin.end(), greater<dist2pt>());
                isaccepte[raymaxmin[0].pt] = true;
```

```
                bool accepte;
                nbpts = 1;
                for(i = 1; i < raymaxmin.size() && nbpts < nb_pf; i++) {
                        if(raymaxmin[i].dist < RAPPORT_MIN) break; // Ne pas accepter les
rapports MAX/MIN sous un certain seuil
                        ptcur = raymaxmin[i].pt;
                        accepte = true;
                        if(seuil > 0.0) { // Ne pas perdre de temps s'il n'y a pas de
seuil
                                // Cette boucle vérifie tous les points plus grands que le
point courant
                                // (Ces points sont stockés dans geodesics[ptcur])
                                for(j = 0; j < geodesics[ptcur].size() && accepte; j++) {
                                        if(isaccepte[geodesics[ptcur][j].pt]) { //
Seulement si le point est déjà accepté
                                                if(geodesics[ptcur][j].dist <= seuil)
                                                        accepte = false;
                                        }
                                }
                                // Cette boucle vérifie tous les points plus petits que le
point courant
                                // (Ces points sont stockés dans geodesics[j] et il faut y
trouver ptcur)
                                for(j = 0; j < ptcur && accepte; j++) {
                                        if(isaccepte[j]) { // Seulement si le point est
déjà accepté
                                                // Vérifier toutes les distances géodésiques
du point j
                                                // pour voir si on y trouve ptcur
                                                for(k = 0; k < geodesics[j].size() &&
accepte; k++) {
                                                        if(geodesics[j][k].pt == ptcur) {
                                                                if(geodesics[j][k].dist <=
seuil)
                                                                        accepte = false;
                                                                break; // ptcur est trouvé,
on peut quitter la boucle
                                                        }
                                                }
                                        }
                                }
                                if(accepte) {
                                        isaccepte[ptcur] = true;
                                        nbpts++;
                                }
                        }
                } else {
                        mexWarnMsgTxt("Il n'existe aucun point fingerprint valide sur cette
surface.");
                }
                // Effacer tous les point fingerprints invalides
                for(i = 0; i < points.size(); i++) {
                        if(!isaccepte[i]) {
                                fingerprints[i].clear();
                        } else {
                                sort(fingerprints[i].begin(),fingerprints[i].end()); // Mettre les
points en ordre pour former le cercle
                                for(j = 0; j < fingerprints[i].size(); j++) {
                                        fingerprints[i][j].z = distcentre[i][fingerprints[i][j].
rayon];
                                }
                        }
```

```
        }
}

// Retour des résultats
void distgeod::getresults(vector<CVector3> &results, vector<vector<ptrayon> > &resultsPF)
{
        results.clear();
        resultsPF.clear();
        mexPrintf("Distances géodésiques\n");
        compute_geodesic_distances();

        mexPrintf("Nombre de fois avec triangles aigus : %i \n",nbtriaigus);
        mexPrintf("Nombre de fois avec triangles obtus : %i \n",nbtriobtus);
        mexPrintf("Nombre de fois avec triangles virtuels : %i \n",nbtrivirts);

        mexPrintf("%i\nProjection 3D-2D\n",clock());
        // Projette les points 3D sur un plan 2D
        projection3D2D();

        mexPrintf("%i\nChoix des meilleurs\n", clock());
        // Choisir des meilleurs point fingerprints
        meilleursPF();
        mexPrintf("Nombre d'angle 1 trop grand : %i\n",nbangle1tropgros);
        mexPrintf("Nombre d'angle N trop grand : %i\n",nbangleNtropgros);
        mexPrintf("Nombre de nombre de points trop petit : %i\n",nbptstroppetit);

        resultsPF = fingerprints;

        /*mexPrintf("%i \n",clock());
        mexPrintf("Rééchantillonnage\n");
        // Refaire un echantillonnage regulier
        reechantillonne(resultsPF);*/

        mexPrintf("%i\nFormattage des distances géodésiques\n", clock());
        // Formatter les données pour qu'elle entre dans un tableau N*3
        int i,j;
        for(i = 0; i < geodesics.size(); i++) {
                for(j = 0; j < geodesics[i].size(); j++) {
                        results.push_back(CVector3(    (REEL)i,
                // Point 1
                                                                                (REEL)
geodesics[i][j].pt,    // Point 2
                                                                                geodesics[i]
[j].dist));            // Distance
                }
        }
}

extern void _main();
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]) {
        int i, j;
        if (nrhs != 7) {
                mexErrMsgTxt("distgeod requires 7 input arguments.");
        }
        if (nlhs > 2) {
                mexErrMsgTxt("distgeod accepts 2 output arguments.");
        }
        // check array dimentions
        if(mxGetN(prhs[0]) != 3){
                mexErrMsgTxt("Points must be NbPoints x 3");
        }
        if(mxGetN(prhs[1]) != 3){
                mexErrMsgTxt("Triangles must be NbTriangles x 3");
```

```
	}
	if(mxGetN(prhs[2]) != 1){
		mexErrMsgTxt("The radius vector must be N x 1");
	}
	if(mxGetN(prhs[3]) != 1){
		mexErrMsgTxt("Maximum distance is a scalar");
	}
	if(mxGetN(prhs[4]) != 1){
		mexErrMsgTxt("Number of points per circle is a scalar");
	}
	if(mxGetN(prhs[5]) != 1){
		mexErrMsgTxt("Number of point fingerprints returned is a scalar");
	}
	if(mxGetN(prhs[6]) != 1){
		mexErrMsgTxt("Minimum distance between 2 chosen points is a scalar");
	}

	// get array pointers
	int nbPoints = mxGetM(prhs[0]);
	int nbTriangles = mxGetM(prhs[1]);
	int nbRayons = mxGetM(prhs[2]);
	double* dPoints = mxGetPr(prhs[0]);		// Points
	double* dTriangles = mxGetPr(prhs[1]);		// Triangles
	double* dRayons = mxGetPr(prhs[2]);		// Rayons des PF
	double dmax = mxGetScalar(prhs[3]);		// dmax
	int nb_ppc = mxGetScalar(prhs[4]);		// Nombre de points par cercle
	int nb_pf = mxGetScalar(prhs[5]);		// Nombre de PF a retourner
	double seuil = mxGetScalar(prhs[6]);		// Nombre de PF a retourner

	vector<CVector3> lPoints;
	vector<triangle> lTriangles;
	vector<CVector3> lNormals;
	vector<REEL> lRayons;
	vector<CVector3> vDistGeod;
	vector<vector<ptrayon> > vFingerPrints;
	vector<SurfaceRegistration::Vertex> srVertices;	// pour utiliser le calcul de
normals prensent dans la classe Mesh
	vector<SurfaceRegistration::Triangle> srTriangles; // idem

	for(i=0; i<nbPoints; i++)	{
		CVector3 currentPoint(dPoints[i], dPoints[i+nbPoints], dPoints
[i+nbPoints*2]);
		lPoints.push_back(currentPoint);
		srVertices.push_back(SurfaceRegistration::Vertex(currentPoint));

	}
	for(i=0; i<nbTriangles; i++)  {
		lTriangles.push_back(triangle(	(int)dTriangles[i]-1,
                                                                        (int)
dTriangles[i+nbTriangles]-1,
                                                                        (int)
dTriangles[i+nbTriangles*2]-1));
		srTriangles.push_back(SurfaceRegistration::Triangle((int)dTriangles[i]-1,
                                                                        (int)
dTriangles[i+nbTriangles]-1,
                                                                        (int)
dTriangles[i+nbTriangles*2]-1));
	}

	SurfaceRegistration::Mesh mesh(srVertices,srTriangles);
	mesh.computeInternals();
	for(i=0; i<nbPoints; i++)
		lNormals.push_back(mesh.getVertex(i).Normal);
```

```
        for(i=0; i<nbRayons; i++)      {
                lRayons.push_back(dRayons[i]);
        }

        mexPrintf("Création de l'objet\n %i\n",clock());
        distgeod calculdg(lPoints, lTriangles, lNormals, lRayons, dmax, nb_ppc, nb_pf,
seuil);
        calculdg.getresults(vDistGeod, vFingerPrints);

        int nbDistGeod = vDistGeod.size();
        int nbFingerPrints = vFingerPrints.size();

        int index = 0;
        for(i=0; i<nbFingerPrints; i++)       index += vFingerPrints[i].size();

        plhs[0] = mxCreateDoubleMatrix(nbDistGeod,3,mxREAL);
        plhs[1] = mxCreateDoubleMatrix(index,5,mxREAL);
        double* outDistGeod = mxGetPr(plhs[0]);
        double* outFingerPrints = mxGetPr(plhs[1]);

        for(i=0; i<nbDistGeod; i++)    {
                outDistGeod[i]                          = vDistGeod[i].x+1; // Ramener en
indices MATLAB
                outDistGeod[nbDistGeod+i]      = vDistGeod[i].y+1;
                outDistGeod[nbDistGeod*2+i]    = vDistGeod[i].z;
        }

        mexPrintf("%i\nReformattage des point fingerprints\n",clock());
        int m = 0;
        for(i=0; i<nbFingerPrints; i++)       {
                for(j=0; j<vFingerPrints[i].size(); j++)      {
                        outFingerPrints[m]                      = i + 1;
                                        // Point courant
                        outFingerPrints[index+m]       = vFingerPrints[i][j].rayon + 1;
        // Numéro du rayon
                        outFingerPrints[2*index+m]     = vFingerPrints[i][j].x;
                // Coordonnée en x
                        outFingerPrints[3*index+m]     = vFingerPrints[i][j].y;
                // Coordonnée en y
                        outFingerPrints[4*index+m]     = vFingerPrints[i][j].z;
                // Décentrage du rayon
                        m++;
                }
        }
        mexPrintf("Fin\n");
}
```